Mining and Linking Crowd-based Software Engineering How-to Screencasts

Parisa Moslehi

A Thesis
In the Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Computer Science) at
Concordia University
Montreal, Quebec, Canada

April 2020
© Parisa Moslehi, 2020

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Parisa Moslehi

Entitled: Mining and Linking Crowd-based Software Engineering How-to Screencasts

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy in Computer Science

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Akshay Kumar Rathore	Chair
Dr. Roberto Lopez	External Examiner
Dr. Jamal Bentahar	External to Program
Dr. Joey Paquet	Examiner
Dr. Emad Shihab	Examiner
Dr. Juergen Rilling	Thesis Supervisor (s)
Dr. Bram Adams	

Approved by Leila Kosseim

Chair of Department or Graduate Program Director

21/04/2020

Date of Defence

Mourad Debbabi

Dean,

Abstract

Mining and Linking Crowd-based Software Engineering How-to Screencasts

Parisa Moslehi, Ph.D.

Concordia University, 2020

In recent years, crowd-based content in the form of screencast videos has gained in popularity among software engineers. Screencasts are viewed and created for different purposes, such as a learning aid, being part of a software project's documentation, or as a general knowledge sharing resource. For organizations to remain competitive in attracting and retaining their workforce, they must adapt to these technological and social changes in software engineering practices.

In this thesis, we propose a novel methodology for mining and integrating crowd-based multi-media content in existing workflows to help provide software engineers of different levels of experience and roles access to a documentation they are familiar with or prefer. As a result, we first aim to gain insights on how a user's background and the task to be performed influence the use of certain documentation media. We focus on tutorial screencasts to identify their important information sources and provide insights on their usage, advantages, and disadvantages from a practitioner's perspective. To that end, we conduct a survey of software engineers. We discuss how software engineers benefit from screencasts as well as challenges they face in using screencasts as project documentation.

Our survey results revealed that screencasts and question and answers sites are among the most popular crowd-based information sources used by software engineers. Also, the level of experience and the role or reason for resorting to a documentation source affects the types of documentation used by software engineers. The results of our survey support our motivation in this thesis and show that for screencasts, high quality content and a narrator are very important components for users.

Unfortunately, the binary format of videos makes analyzing video content difficult. As a result, dissecting and filtering multimedia information based on its relevance to a given project is an inherently difficult task. Therefore, it is necessary to provide automated approaches for mining and linking this crowd-based multimedia documentation to their relevant software artifacts. In this thesis, we apply LDA-based (Latent Dirichlet Allocation) mining approaches that take as input a set of screencast artifacts, such as GUI (Graphical User Interface) text (labels) and spoken words, to perform information extraction and, therefore, increase the availability of both textual and multimedia documentation for various stakeholders of a software product. For example, this allows screencasts to be linked to other software artifacts such as source code to help software developers/maintainers have access to the implementation details of an application feature.

We also present applications of our proposed methodology that include: 1) an LDA-based mining approach that extracts use case scenarios in text format from screencasts, 2) an LDA-based approach that links screencasts to their relevant artifacts (e.g., source code), and 3) a Semantic Web-based approach to establish direct links between vulnerability exploitation screencasts and their relevant vulnerability descriptions in the National Vulnerability Database (NVD) and indirectly link screencasts to their relevant Maven dependencies. To evaluate the applicability of the proposed approach, we report on empirical case studies conducted on existing screencasts that describe different use case scenarios of the WordPress and Firefox open source applications or vulnerability exploitation scenarios.

To my parents

ACKNOWLEDGEMENTS

I would never have been able to finish my dissertation without the guidance of my advisors Dr. Juergen Rilling and Dr. Bram Adams who accepted me as their PhD student and offered me their mentorship. My advisors always provided me with excellent guidance and were available for having insightful discussions and asking challenging questions to improve this work. I would like to express my deepest appreciation to them for their endless care, patience, constructive feedback, and for providing me with an excellent atmosphere for doing research and patiently editing my writings. Their own passion, hard work, and unflinching courage has always inspired me to do more, overcome many difficulties and learn a lot. Their guidance indeed has made this a thoughtful and rewarding journey.

I would like to thank my examining committee including Dr. Jamal Bentahar, Dr. Joey Paquet, Dr. Emad Shihab, and Dr. Roberto Lopez (External) for their valuable feedback on my research that helped me significantly improve this work.

I also had a great pleasure of working with the Ambient Software Evolution Group (ASEG) members at Concordia University and the members of the lab on Maintenance, Construction and Intelligence of Software (MCIS) at Polytechnique Montreal. I would like to express my deep gratitude to an amazing friend and colleague Dr. Ellis Emmanuel Eghan for his helpful discussions and great collaboration which led to publishing a new research work.

I would like to extend my sincere thanks to my friends Maysam Mokarian and Dr. Nafiseh Barzigar who never wavered in their support and always believed in me and encouraged me to pursue my dreams. I thank all my friends who always were there for me and gave me the necessary distractions from my research and made my PhD studies memorable.

I cannot begin to express my thanks to my dear family for their care, unconditional love, and sacrifice. I am deeply indebted to my lovely parents Aghdas and Hossein, my sister and role model Fariba, my brothers, my brother and sister in laws, my niece Ladan, and my nephews who always showed faith in me and encouraged me to follow my dreams in my life. They always accepted and supported my decisions in my journey of learning form the first day of the elementary school and prayed for my success.

Contribution of Authors

The work in Chapter 9 is the result of a collaborative work with Dr. Ellis Emmanuel Eghan (the former member of ASEG lab). My main contribution towards the work in this chapter is on extracting information from screencasts, mining screencasts, and designing and creating the video ontology (VIDONT).

Table of Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface
CMS	Content Management System
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities Exposure
CWE	Common Weakness Enumeration
DL	Description Logic
GUI	Graphical User Interface
HD	High Definition
IDE	Integrated Development Environment
LDA	Latent Dirichlet Allocation
LSI	Latent Semantic Indexing
MSR	Mining Software Repositories
NVD	National Vulnerability Database
OCR	Optical Character Recognition
OMG	The Object Management Group
OWASP	The Open Web Application Security Project
OWL	The Web Ontology Language
RDF	Resource Description Framework
SBSON	The Software Build System Ontology
SE	Software Engineering
SEON	The Software Engineering Ontologies
SEVONT	The Security Vulnerability Ontology
SPARQL	Simple Protocol and RDF Query Language
STT	Speech To Text
SV-AF	Security Vulnerability Analysis Framework
SVDB	Security Vulnerability Database
SWRL	Semantic Web Rule Language
VIDONT	Video Ontology

1 Introduction

The last three decades have been characterized by new technologies (e.g., collaborative development, social media) that not only have major effects on peoples' daily lives but also on workplaces and organizations. These technological and social changes were not instant, rather they evolved slowly, with various age groups (a.k.a. generations) of the population being exposed and immersed differently within these changed new environments. Organizations have to adapt to these changes to provide work environments and a *media ecology* [1] that not only supports complementary tools of communication but also allows organizations to take full advantage of these skill changes in the workforce.

At the same time, traditional well-formed and structured software documentation and processes have been replaced with new, more agile software processes reducing the amount of documentation explicitly generated for projects. This transition of traditional software engineering workflows and best practices has been further accelerated through the use of social media and crowd-based documentation while having a generation of younger developers in the workforce, who were immersed in these new technologies while growing up.

The wide use of social media channels (e.g., e-mail, chat, vlog) for interaction and collaboration among people has also affected traditional processes in the software engineering community. Developers are now commonly using these social media channels to share knowledge and interactively learn from one another (e.g., how to perform certain tasks or to improve their work [1]–[5]). Key to this widespread social and multimedia environments adoption is that they are readily available and provide cheap and fast communication and information distribution [1].

As part of these new communication channels, new media resources such as images (e.g., captured from whiteboards), Wikis, Q&A (Question and Answer) repositories (e.g., Stack Overflow), and video repositories (e.g., YouTube or Instagram) [2], [6] have started to become part of current work practices. These resources are often created by many and used by many [6], with contributors being motivated by the fact that they can gain an online reputation and also learn more through the process of teaching others. Such crowd-based documentation or resources are considered a subset of what is known as *public good* [7], which refers to content that is socially generated by the community or within organizations (e.g., documentation, code, activities, etc.).

The adoption of social media tools within the software engineering community has been investigated by Black et al. [5] and Storey et al. [1]. Black et al. [5] found that social media tools are useful in a sense that they improve the speed of communication and facilitate communication among colleagues. Storey et al. [1] found that while traditional and face-to-face communication is still an important means of communication among developers, social media channels are considered as valuable tools for collaboration and communication within a software engineering community of practice. This resulted in the emerging notion of *social programmers* [8] who leverage social media tools to participate and contribute to the creation of crowd-based resources. However, this requires a level of *social media literacy* [8] to be able to effectively leverage the tools and interact with the community.

One type of crowd-based multimedia documents that has gained in popularity, are screen-captured videos (i.e., screencasts). Screencasts deliver content in the form of audio (through a narrator), video (images), and textual (meta) data (e.g., subtitles, title, description, publish date). While other text-based documentation such as blog posts, question and answers, and traditional software artifacts are used to share explicit knowledge or facts about an application [9], screencasts are used for sharing tacit knowledge [9] that is in people's heads [7]. Screencasts can be created for different purposes, such as: explaining how to use a new feature, providing step-by-step instructions for workarounds to a given problem, or to demonstrate security issues that may result from known security vulnerabilities [10].

While some research [1], [4], [6], [8], [11] investigated the use of social media in general in software engineering, little research exists on understanding the **creation** and **usage** of different types of **screencasts** for software engineering. Among the earliest work is MacLeod et al. [9], who conducted a user study to gain insights on how and why software developers use programming screencasts for documenting software. In their study they focused on programming tutorial screencasts that contain live coding. They observed that developers **create** screencasts to generate and maintain an online identity, to learn by teaching others, and to share their knowledge with the community. Also, their research shows that one approach of screencast creators in creating code documentation is "*doing by showing.*" In this approach they demonstrate the execution of a use case in the program while performing live coding to help people understand the data flow and expected output of the program. Such mapping of execution to code helps the audience to also better understand where in the code certain program features are implemented. While their research was on the **creation** of crowd-based screencasts for programming purposes, they also highlight the need for understanding how and why screencasts are **used** by developers and software engineers.

In this thesis, we study the adoption or **usage** of crowd-based multimedia documentation as an information source in the software engineering domain and introduce a novel methodology for mining crowd-based software engineering tutorial screencasts and linking them to other software artifacts to overcome existing challenges of adopting screencasts in software engineering (see Chapter 2). Among the challenges related to the use of social media and its content for software engineering are: a) trustworthiness, quality, and reliability of the media content; b) problems in finding and retrieving information related to a specific version of a project; c) change impact analysis and updating information; and d) lack of traceability links between multiple media channels and other software artifacts such as design, algorithms, and programming artifacts.

Addressing these challenges by mining and integrating social media documentation (e.g., screencasts on YouTube) with other software artifacts (e.g., source code) will improve the acceptance and use of these media resources in software engineering. Furthermore, automatically mining and linking screencasts that demonstrate how to use a software application's features and their source code implementation will facilitate and speed up software maintenance activities. Examples of such activities can be adding/removing/modifying a GUI feature and updating its relevant documentation (e.g., screencasts) or fixing bugs that are reproduced and reported via a screencast or requirements that are mentioned while demonstrating a feature in a screencast.

While the MSR (Mining Software Repository) community has, over the last decade, started to mine unstructured repositories such as mailing lists, Q&A sites, and Wiki pages by analyzing their content, only limited work exists on mining, linking, and analyzing the content of video repositories. Aggregating social media channels (e.g., GitHub) or integrating them into the IDEs (Integrated Development Environment) has shown to be useful in improving the reliability and the use of socially produced content and code artifacts [1]. Ponzanelli et al. [3], [12], as well as other researchers [10], [13]–[18] have addressed the challenges of integrating crowd-based programming screencasts with other software artifacts, including Q&A sites such as Stack Overflow. However, in their work they tackle the problem of linking programming screencasts that contain source code (and not GUI) with other artifacts, while linking GUI-based screencasts to their relevant artifacts is another challenge that has yet to be addressed.

1.1 Our Thesis

The adoption of social media resources and mining crowd-based multi-media documentation is still an emerging research field [2], [9]. Based on the observations presented in the previous section, we can define our research thesis as follows:

Mining and linking crowd-based screencasts that showcase GUI features of an application is feasible and allows to support a variety of software maintenance and development tasks.

In what follows, we provide a summary of our research contributions.

1.2 Summary of Research Contributions

In this thesis, we address the challenges of helping different project stakeholders from different backgrounds, expertise, and social media literacy to have ubiquitous access to screencasts' content as an information source as well as the possibility of tracing down other software artifacts from screencasts. Towards this end, we identify and define the primary contributions of our research, stated as follows:

- **Conducting a survey on the adoption of multi-media documentation (Chapter 5)**
    - As part of this research contribution, we conducted a survey to identify the source(s) of documentation that are preferred by software engineers with different characteristics, and the context in which they use this documentation (e.g., help with a particular software engineering task, to improve their current skills).
    - Through the study, we identify components or information sources of a tutorial screencast that are important to locate screencasts that are useful and clearly describe a specific task.

- We provide insights on how software engineers use how-to tutorial screencasts as a software artifact and their role as a source of documentation in the software engineering domain.

The results of the survey support the need for designing and evaluating a methodology for mining and linking crowd-based software engineering tutorial screencasts.

- **A methodology for mining and linking crowd-based screencasts (Chapter 6)**
    - As part of this research contribution, we propose a new methodology that leverages the high-level information that exists in both audio (i.e., speech) and visual cues (i.e., GUI) from screencasts to link them to other relevant software artifacts (e.g., source code).
    - As part of our proposed methodology, we extract use case scenarios from the speech component of screencasts and show how the speech content can help identify screencasts that are the most relevant to a certain task.

The following contribution explores the use of our methodology for mining and linking crowd-based documentation to support software maintenance or development tasks. As our survey responses also confirm, such methodology is needed to provide high quality and reliable complementary documentation that can help improve the interpretation of screencast content as well as to fulfill different preferences in the type of documentation to be used for understanding how to perform a task.

- **Applying the proposed methodology in software engineering contexts**
    - Mining crowd-based speech documentation (Chapter 7)
    - Feature location using screencasts (Chapter 8)
    - Integrating screencasts with security advisories using the Semantic Web (Chapter 9)

1.3 Related Publications

Applications of the proposed methodology in this thesis have been published in the following papers:

Submitted:

- **Parisa Moslehi**, Juergen Rilling, and Bram Adams, (2020). "Adoption of crowd-based software engineering tutorial screencasts," *Journal of Systems and Software* (Submitted)

Accepted:

- **Parisa Moslehi**, Bram Adams, and Juergen Rilling, (2020). "A feature location approach for mapping application features extracted from crowd-based screencasts to source code," *Empirical Software Engineering* (Accepted for publication).

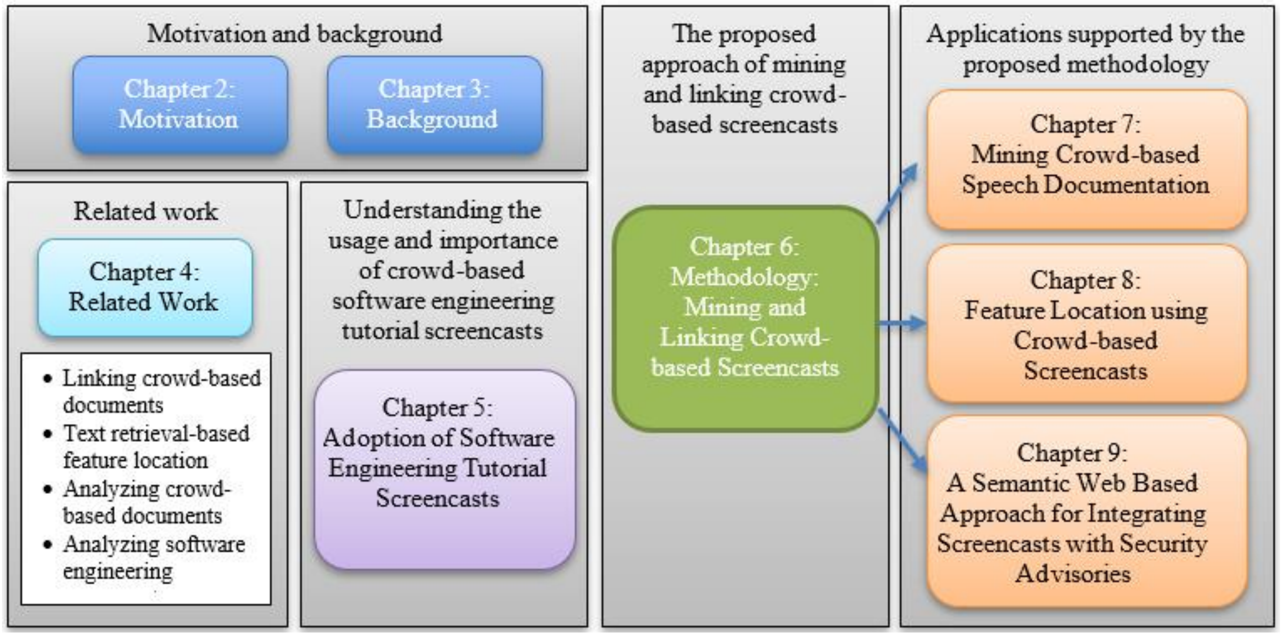
Published:

- Ellis E. Eghan, **Parisa Moslehi**, Juergen Rilling, and Bram Adams, "The missing link – A Semantic Web-based approach for integrating screencasts with security advisories," *Information and Software Technology*, vol. 117, pp. 106197, 2020.
- **Parisa Moslehi**, Bram Adams, and Juergen Rilling, "Feature location using crowd-based screencasts," in *Proceedings of the 15th International Conference on Mining Software Repositories, MSR*, Gothenburg, Sweden, 2018, pp. 192-202.
- **Parisa Moslehi**, Bram Adams, and Juergen Rilling, "On mining crowd-based speech documentation," in *Proceedings of the 13th Working Conference on Mining Software Repositories, MSR*, Austin, Texas, USA, 2016, pp. 259-268.

1.4 Thesis Organization

The remainder of this thesis is structured as shown in Figure 1-1.

In the next chapter, we describe our motivation. Chapter 3 covers the background related to the research. We give a brief introduction to crowd-based multimedia documentation, topic modeling, traceability and feature location, and the Semantic Web that we used in our approaches. Chapter 4 covers related work, in which we describe and contrast similar attempts in analyzing and linking crowd-based documentation, and text retrieval-based feature location. Chapter 5 presents our survey on the adoption of crowd-based multimedia documentation, which provides the basis of our thesis. Chapter 6 explains our proposed methodology of mining and linking crowd-based screencasts. Applications of our proposed methodology and their experiments and results are presented in Chapter 7: Mining crowd-based speech documentation, Chapter 8: Feature location using crowd-based screencasts, Chapter 9: A Semantic Web-based approach for integrating screencasts with security advisories. Finally, we present conclusions and future opportunities in Chapter 10.


Motivation and background
Chapter 2: Motivation
Chapter 3: Background
Related work
Chapter 4: Related Work
• Linking crowd-based documents
• Text retrieval-based feature location
• Analyzing crowd-based documents
• Analyzing software engineering
Understanding the usage and importance of crowd-based software engineering tutorial screencasts
Chapter 5: Adoption of Software Engineering Tutorial Screencasts
The proposed approach of mining and linking crowd-based screencasts
Chapter 6: Methodology: Mining and Linking Crowd-based Screencasts
Applications supported by the proposed methodology
Chapter 7: Mining Crowd-based Speech Documentation
Chapter 8: Feature Location using Crowd-based Screencasts
Chapter 9: A Semantic Web Based Approach for Integrating Screencasts with Security Advisories
Figure 1-1. Overview of the thesis content

2 Motivation

A significant body of work [19]–[26] exists on categorizing different generations of the population based on their signature products, technologies, learning, and communication preferences. Gen X, also referred to as the generation of digital immigrants, are people who did not grow up with digital media during their youth (instead predominantly made use of printed documents) but learned to adapt to these new technologies and to the use of digital media (e.g., e-mail) as a communication medium. Gen Y, also known as millennials[1] and the generation of tech-savvy or digital natives, were exposed to digital technologies early in their lives (either as a child or young adult). Gen Y is known for being visual learners who search online resources for hands-on information rather than using textbooks; a generation that prefers image- and video-based communication and social networks over textual documents.

People of Gen Z, often also referred to as the generation of technoholics, have since their childhood depended on the use of electronic devices and digital content to manage their lives. Ubiquitous computing plays an increasing role for this generation, with boundaries between individual devices disappearing, and data mining and context awareness (e.g., location awareness), which enable services (e.g., recommendations and search results on the Internet) that are tailored to each user's needs, as a growing aspect of their daily lives [27]. Together with generation Y, generation Z is part of the emergence of a participatory culture [8] and have adopted social media into their daily workflows to communicate and collaborate with others or learn from one another. As a result, crowd-based documentations are typically created and consumed by Gen Y and Z users.

Existing differences in the use of technology among generations will partly disappear as a technology matures and its use spreads further across the population. As a result, organizations have started to recognize that their traditional processes and workflows no longer match these new learning and work habits of the younger workforce [22] and have started to adapt their work environments to more closely match the new skillsets and learning styles. Providing such a work

[1] https://en.wikipedia.org/wiki/Millennials

environment has become an essential aspect in not only improving productivity but also in attracting and retaining future employees.

For example, depending on their level of familiarity with digital technologies of the 21st century [22] (or their social media literacy [8]), software developers will rely on resources they are familiar with to support or complete their daily tasks or document their work. Educational systems have already embraced these changes in technologies and learning styles in our society by adopting existing and introducing new learning and teaching approaches to accommodate these changes in learning behavior. An example of such new learning styles is blended-learning (a.k.a., hybrid learning and mixed-mode learning that "blends text-based asynchronous Internet technology with face-to-face learning") [28], which has emerged as an integral part in teaching learners who are more familiar with the new technologies (i.e., digital natives).

Since the majority of current developers are part of Gen Y and Gen Z, generations who are visual learners and prefer digital content over textual documentation, screencasts have become widely accepted in the open source community, to a point where open source projects have started to make how-to screencasts an integrated part of their documentation. For example, WordPress[2] encourages its users and contributors to create new how-to videos[3] that describe certain use case scenarios or features. Also, users have started to enrich their bug reports with screencasts[4] to demonstrate exactly how to reproduce a bug. Such integration of screencasts into software engineering workflows by linking them to other software artifacts has the potential to improve the performance of software developers and maintainers.

Given the increasing role of crowd-based multimedia resources in practice and the challenges (discussed later in this chapter) that exist in their usage, we are motivated to gain a better understanding of the importance of using such how-to screencasts in a software engineer's daily work. We argue that mining crowd-based multi-media content and its integration in existing software engineering workflows can help provide younger or less experienced developers with easier access to documentation they are more familiar with.

[2] https://wordpress.com/
[3] https://en.support.wordpress.com/video-tutorials/
[4] https://youtu.be/Am9SNUhSz4w

In many open source projects or agile development processes, the official system documentation is incomplete, inconsistent, or does not provide clear enough examples. Developers and maintainers, therefore, typically resort to an information resource they are familiar with on the Internet (i.e., social media channels) for additional help or information needs [6], [2]. These online resources are often created by users who are not directly involved in the development of the actual product (i.e., the crowd). Such crowd-based documentation is presented in different formats such as multimedia or informal textual documentation, describing a particular feature, how to solve a particular problem, teaching a programming language, how to fix a bug, etc. [29].

As a result, the overall goal of this thesis is to understand the usage of crowd-based multimedia documentation that is publicly available on social media (e.g., YouTube) and to propose mining approaches that leverage crowd-based multimedia content and more specifically screencasts (e.g., speech and image frames) to perform information extraction or establish traceability links to other software artifacts. This will help make screencasts' content available to various stakeholders of a software product. Improving the accessibility to such crowd-based documentation is one approach to close a growing gap in the use of multimedia documentation among different software developers and maintainers caused by cultural, social, and educational changes (i.e., how to keep the Gen X workforce up-to-date with crowd knowledge that is produced by Gen Y or Z, using the documentation format they are most familiar with?).

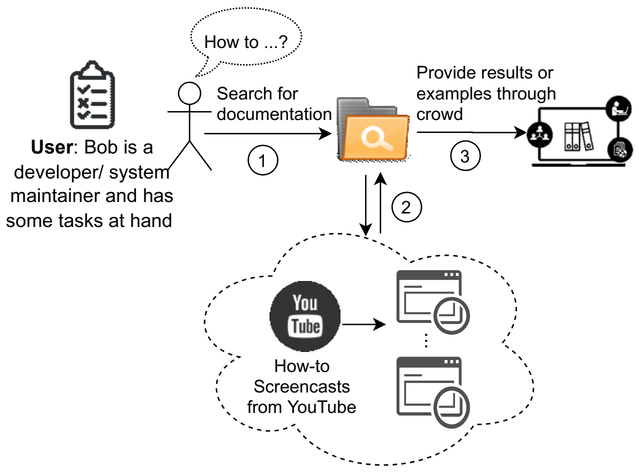

How to ...?
User: Bob is a developer/ system maintainer and has some tasks at hand
Search for documentation
1
2
Provide results or examples through crowd
3
How-to Screencasts from YouTube

Figure 2-1. How mining screencasts' content helps WordPress users or developers.

Figure 2-1 shows an example of a fictional web developer named Bob, an open source enthusiast, who is new to WordPress, a content management system. As part of his contribution, Bob wants to modify a comment moderation feature in the source code of the Akismet WordPress plugin for his personal project. Bob, who is unfamiliar with the workflow and innards of the plugin, first resorts to the available software documentation to understand the high-level context and features the plug-in provides to users. However, Bob is unable to find any relevant documentation describing these features and their (sequential) interaction in the existing project documentation. He therefore decides to search the Internet for how-to videos and blogs describing the plugin features. Such documentation, especially how-to screencasts, by many developers are considered to be more intuitive since they visually and dynamically can demonstrate workflows compared to written documentation [2].

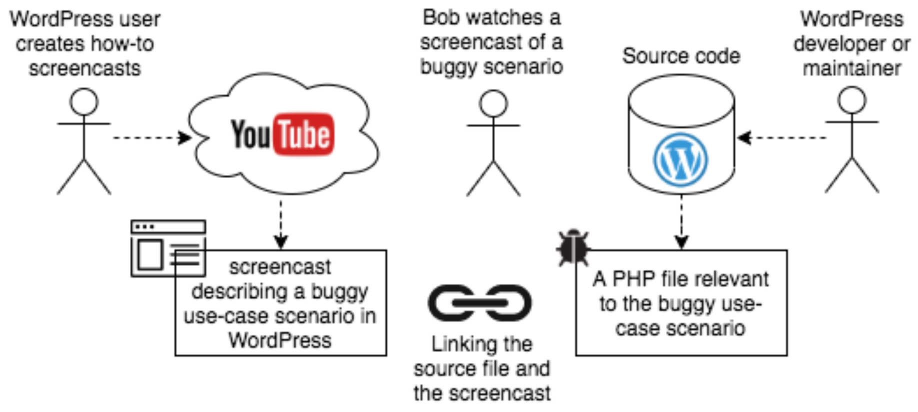

WordPress user creates how-to screencasts
You Tube
Bob watches a screencast of a buggy scenario
Source code
WordPress developer or maintainer
screencast describing a buggy use-case scenario in WordPress
Linking the source file and the screencast
A PHP file relevant to the buggy use-case scenario

Figure 2-2. How linking source code features to screencasts helps WordPress users or developers.

While watching the how-to video of a user, Bob notices that the narrator mentions a bug when setting up the plug-in in WordPress dashboard. Bob decides **to contribute to the WordPress plugin (Figure 2-2)** by fixing the bug that was shown in the screencast. However, Bob, who is still unfamiliar with the implementation details of the plug-in, finds it challenging to locate the implementation of the buggy feature in the source code. Furthermore, users may also create a bug report that is enriched by a screencast that clearly showcases the steps of reproducing a bug[5] on the GUI of an application or provide workarounds for an issue[6]. Developers then must locate the source code artifacts that are relevant to the bug to fix the reported issue.

As these motivating examples have illustrated, software developers and maintainers might rely on screencasts for different reasons (e.g., reproducing a bug, vulnerability exploitation, and etc.). This diversity in screencast use as a source of software documentation creates several challenges

[5] https://www.reddit.com/r/firefox/comments/4fq1g0/firefox_ui_bug/
[6] https://youtu.be/CIvTVvFTWDA

for users and creators of screencasts [2]. We categorized these challenges, which are also identified in our survey (see Chapter 5), as follows:

- **Popularity:** What group of developers often refer to screencasts to learn something new? How-to videos and visual documentation are mostly accepted and used as a source of information by visual learners or beginners (i.e., Gen Y and Z software engineers or less experienced ones), while textual documentation is more prevalent among more skillful or experienced software engineers.
- **Relevance**: How closely a screencast addresses the task at hand? Although screencasts and crowd-based documentation contain useful instructions on how to accomplish a specific task, developers may spend significant time trying to identify what video to watch. Developers often must rely on manual browsing through videos to see whether the video contains content that is relevant to their needs or to skip nonrelevant parts of videos. Existing search engines rely on the indexing of user-provided titles and descriptions, or words recovered from automatic speech-to-text, and metadata such as number of likes/dislikes and views, but do not consider the actual content (semantics) in their search and ranking.
- **Quality:** How clearly and thoroughly a video presents a task? Even if a video is relevant, its content and technical quality might vary and is unknown to the users before watching it. For example, a video might not cover a complete use case at hand, or video or speech could contain noise.
- **Quantity:** How many videos are created that cover the same topic? Depending on the popularity of the topic, many videos might exist, which vary significantly in their length and content. Some videos may contain more details about the topic, while others may stick to the point or only cover the basics of the topic.
- **Traceability:** What are the software artifacts that are relevant to the scenario that is presented in a screencast? While screencasts demonstrate buggy scenarios or workarounds, developers who want to fully comprehend such bugs or integrate the described workarounds into the application still need to manually trace the screencast content to the application’s source code. This process will be time-consuming especially if the developer is unfamiliar with the project.

In this thesis, we aim to address these challenges by exploring multi-media documentation and proposing an approach of mining and linking multimedia content that provides the basis for solutions that lead to having an optimized software documentation system, in which textual and multi-media documentation is available and traceable.

3 Background

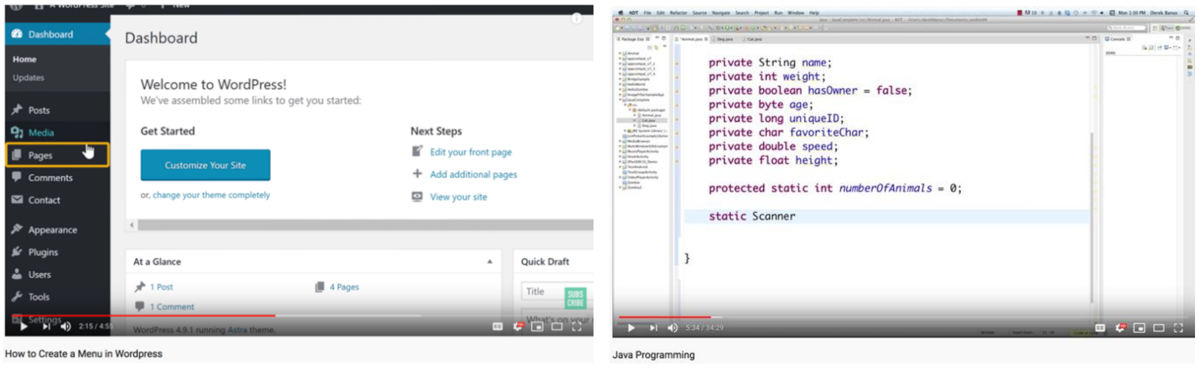
The proposed research is based on several concepts and techniques from different sub-domains within computer science, including crowd-based multi-media documentation, topic modeling, traceability and feature location, and ontologies and Semantic Web. In this chapter, we provide a brief introduction to the underlying concepts and techniques used in this thesis. If you are already familiar with these concepts, you can safely move on to the next chapter.

3.1 Crowd-based Multi-Media Documentation and Screencasts

Software users and developers use the Internet to search for informal documentation that provides help and support for specific tasks. These types of documentation that are created by many and viewed by many are called "crowd-based documents" [6], [18]. Crowd-based documents can be categorized in two formats of textual (e.g., Wikis, emails, Q&As[7]) and multimedia (e.g., images, podcasts, screencasts[8]) documents.

New media types (e.g., images captured from whiteboards), wikis, Q&A repositories (e.g., Stack Overflow) and video repositories (e.g., YouTube or Instagram) [2], [6] have started to become part of current work practices. Such crowd-based documentation or resources are a subset of what is known as *public good* [7] that refers to the content that is socially generated by the community (e.g., documentation, code, activities, etc.). Such digital and multimedia content has become a main stream resource for practitioners while developing and maintaining software systems.

[7] Portals such as https://www.wikipedia.org/ and https://stackoverflow.com/ contain crowd-based textual documentation.
[8] Portals such as https://commons.wikimedia.org/wiki/Main_Page and https://www.youtube.com/ contain crowd-based multimedia documents.



```
private String name;
private int weight;
private boolean hasOwner = false;
private byte age;
private long uniqueID;
private char favoriteChar;
private double speed;
private float height;

protected static int numberOfAnimals = 0;

static Scanner

}
```

(a) (b)

Figure 3-1. A screencast that showcases applications' GUI features (a). A screencast that showcases programming features (b).

Multimedia documents, and more specifically screencasts, differ from formal and textual documents. Screencasts are "digitally recorded playback of computer screen output that often contains audio narration" [30]. In the software engineering domain, how-to screencasts are movies of software that demonstrate application features, programming features, or software processes (Figure 3-1) while, if available, a narrator verbally describes them [9]. The motivation behind creating such documents is for the document creators to gain an online reputation as well as to learn a new subject better and improve their skills by teaching them to others [2].

Sugar et al. identified three structural elements that are common among screencasts: a) "*bumpers*", which correspond to the beginning and ending of screencasts where the instructor introduces herself, greets the audience, or finishes the tutorial (e.g., general opening/ closing remarks or inviting the audience to subscribe to the channel), b) "*Screen movement*" can be dynamic in the sense that the screen moves so that the cursor remains in the center of the frame, or static where the cursor moves within a frame, c) "*Narration*", i.e., explicit description of the content according to which the narrator 1) explicitly describes whatever she is doing on the screen or 2) provides an implicit description (i.e., a high level statement) which refers to what she is going to do or the steps she follows on the screen that can visually be followed.

In what follows, we list some characteristics that also make it challenging for screencasts to be analyzed and integrated with other types of software artifacts.

- **The abstraction level:** As discussed above, one structural element of a screencast can be "*Narration*". As a result, content of a screencast is typically delivered by a narrator who is describing verbally (through speech) and visually (by demonstrating) the screencast content. Such a screencast content delivery differs significantly from other software artifacts, where content is often represented in structured or semi-structured textual formats that simplify the automated content processing and analysis. Furthermore, even technical screencasts tend to be at a higher level of abstraction, only sparsely providing low-level implementation details of the demonstrated features, which makes it challenging for the developers to locate other software artifacts (e.g., code fragments) that are relevant to a particular screencast.

- **Dynamic vs. Static:** Screencasts are *dynamic* by nature, i.e., every couple of image frames the screen changes as the narrator is manipulating the GUI of the demonstrated software application [3]. In contrast, there are other crosscutting software artifacts (e.g., source code, emails, wiki) that are static and relevant to the demonstrated version of the software application. This crosscutting makes it inherently difficult to manually identify and link such artifacts relevant to the dynamic content of a screencast shown at some point in a screencast.
- **Information resources:** Screencasts can contain different information resources (e.g., images, speech, captions, sequence of GUI events, user actions, metadata) [2], which differ significantly from those found in software artifacts such as source code (e.g., language-specific syntax, comments, function names, identifiers, file names, string literals).

3.2 Topic Modeling

Topic models provide statistical information related to sets of words ("topics") that occur together often enough to represent a semantic relation [31]. For example, in a newspaper the "sports" topic consists of sports-related words that are semantically related and co-occur across most of the sports-related articles. Topic models can be used in automatic indexing, searching, classifying, and structuring a large corpus of text [32]. Generally, topic modeling has been used in different software engineering tasks such as document clustering [33], [34], feature location [35], [36], bug prediction [37], source code evolution [38], traceability [39], [40], and search [41], [42]. The repositories whose data have been used as input for topic modeling include source code, email, requirements and design documents, logs, and bug reports [32].

Latent Dirichlet Allocation (LDA) is a probabilistic topic model proposed by Blei et al. [43]. LDA is a fully generative model that works in a way that assumes that the corpus contains a set of

K topics. The topics are corpus-wide, which means that each document of the corpus contains one or more of the global set of topics and one topic's distribution may be dominant in the document (e.g., if $K = 3$, a document can contain A% of topic 1, B% of topic 2, and C% of topic 3, where A% + B% + C% = 100%). Also, each term in the entire corpus vocabulary can be contained in more than one topic and each term in a document originates from one single topic. As a result, using LDA, each document can be described by a distribution of topics and each topic can be described by a distribution of words. This knowledge can then be used on a new document to find which category the document belongs to.

We also can specify the gram size for topic modeling. For instance, bigram topic models split the text into word groups of length 2 (e.g., 'add_new', 'blog_post', 'search_engine', 'remove_password', etc.). Using bigram topic modeling, the model explicitly encodes word order and context by exploiting that word *A* was immediately succeeded by word *B* [44] after having removed stop words. Compared to unigram topic models, bigram topic models are more useful for understanding the semantics of a text, since the meaning of some words may be affected by their precedent or subsequent word in a sentence (e.g., 'search' and 'engine' vs. 'search_engine').

The most important benefit of LDA is that, since topics are corpus-wide and generated based on all available documents in the corpus, topics for newly added documents can easily be inferred. Also, LDA overcomes the statistical shortcomings of LSI, such as the assumption that the term counts in the corpus follow a Gaussian distribution [45]. In addition, LDA generates human-readable topics that can be used and interpreted easily. As a result, LDA has become a popular topic modeling approach. Chen et al. [32], provide an overview of other variants of LDA (e.g., Hierarchical Topic Models (HLDA) [46], Supervised Topic Models (sLDA) [47], Labeled LDA (LLDA) [48], etc.). However, LDA is the most popular algorithm that is used in the software engineering domain.

3.3 Traceability and Feature Location

Software traceability has been widely recognized as an important quality factor of well-engineered software systems [49]. Software and system traceability can be defined as the ability to establish links between related software artifacts to maintain safe and correct operation of critical software systems [50]. Feature location is a form of traceability that automatically discovers the source code artifacts that implement a certain feature of a software application [32]. It aims to find the starting point (seed) in the source code that corresponds to a system functionality to guide manual exploration [51]. Using such a seed will reduce the effort required by a software maintainer to locate the parts of the code relevant to a particular feature.

Software traceability can help stakeholders to discover discrepancies and inconsistencies between requirements and their implementation, assessing if software requirements are completely covered in the implementation, and is typically required to receive certificates of assurance [52], which are certificates that ensure the software conforms to user requirements or other policies [53].

For open-source projects, it has been shown that the requirements analysis and requirements traceability of a project is very different from that of traditional approaches in software engineering [54]. There are different types of informal resources that capture requirements and documentation of a project, such as issue trackers, emails, source control repositories, and screen-captured videos. To reduce maintenance effort, it is essential to keep such project documents consistent with the current state of the source code by being able to uncover traceability links between these artifacts [55].

3.4 Ontologies and Semantic Web

The *Semantic Web* has been defined by Berners-Lee et al. as "an extension of the Web, in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [56]. It forms a Web from documents to data, where data should be accessed using the general Web architecture (e.g., URIs). Using this Semantic Web infrastructure allows data to be linked, just as documents (or portions of documents) are already, allowing data to be shared and reused across application, enterprise, and community boundaries. In a Semantic Web, data can be processed by computers as well as by humans, including inferring new relationships among pieces of data.

HTML, for instance, can present information in terms of how information is displayed by machines to the user, but it lacks the necessary semantics to allow for further machine interpretation of the displayed facts in terms of their meanings. The Semantic Web overcomes this limitation by adding semantics to the information, making information machine processable and linkable. For example, a YouTube video “File Upload in Java Servlet” has been created by the YouTube contributor “Telusko”. Analyzing the HTML source of the web page would allow us to identify that a text string “Telusko” exists, but in contrast to the Semantic Web, HTML does not allow us to reason or associate that “Telusko” corresponds to the author of the video.

For machines to understand and reason about knowledge, this knowledge needs to be represented in a well-defined, machine-readable language. Ontologies provide a formal and explicit way to specify concepts and relationships in a domain of discourse. The Semantic Web uses the Resource Description Framework (RDF) as its underlying data model to formalize the meta-data of real-world resources as subject-predicate-object triples, which are stored in triple-stores. A resource in Semantic Web can be anything: a person, project, software, a security bug, etc. Triple-stores are Database Management Systems (DBMS) for data modeled using RDF. Unlike Relational Database Management Systems (RDBMS), which store data in relations (or tables) and are queried using SQL, triple-stores store RDF triples and are queried using SPARQL [56]. The RDF data-model is domain independent, and users define ontologies using an ontology definition language.

The *Web Ontology Language (OWL)* [57] is an example of such a definition language and has been standardized by the W3C[9]. It supports the creation of machine-understandable information to enable Web resources to be automatically processed and integrated. The OWL-DL sub-language, is based on Description Logics (DLs) [58]. DL is a logic-based formalism using predicate calculus to define facts that can formally describe a domain. Therefore, DLs are a set of axioms called a TBox (e.g., $Doctor \sqsubseteq Person$) and set of facts called ABox (e.g., {Parent(John), hasChild(John, Mary)}). Both TBox and ABox form a Knowledge Base (KB) and are often written $K = \prec T, A \succ$. The RDF data-model forms a graph where nodes (subject, object) are connected through edges (predicates). The SPARQL query language [59] is used to retrieve information from RDF data-model graphs.

Ontologies vs. Models. A model is “an abstraction that represents some view on reality, necessarily omitting details, and for a specific purpose” [60]. In SE, ontologies and models try to address the same problems (representing the software complexity abstractly) but from very

[9] https://www.w3.org/

different perspectives. The differences between ontologies and models often result in different artifacts, uses, and possibilities. For example, modern SE practices advise developers to look for components that already exist when implementing functionality, since reuse can avoid rework, save money, and improve the overall system quality [61]. In this example, ontologies can provide clear advantages over models in integrating information that normally resides isolated in several separate component descriptions. Furthermore, models (e.g., UML) rely on the closed world assumption, while ontologies (e.g., OWL) support open-world semantics. OWL, an example ontology language, is a "computational logic-based language" that supports full algorithmic decidability in its OWL-DL (description logic) variant. It is not possible to use algorithms supported by OWL (e.g., subsumption) for modeling languages due to their different semantics. Additional differences between ontologies and models are reported and discussed elsewhere [62].

3.5 Chapter Summary

In this chapter, we provided background information for core concepts and algorithms that are used throughout our research. In the next chapter, we review research that is directly related to our research area and identify current research gaps.

4 Related Work

In this chapter, we provide an overview of a selection of closely related works to our proposed research along with a discussion on how our proposed research fill the current research gaps. We first describe existing works related to linking and analyzing crowd-based documents in general, followed by a review of text retrieval-based feature location. We then present current work in linking and analyzing software engineering screencasts.

4.1 Linking and Analyzing Crowd-based Documents

A significant body of work exists that attempts to establish traceability links between crowd-based documents such as Q&A sites (e.g., [4], [29], [63]) and source code artifacts. For example, Jiau and Yang [63] measured the inequality of crowdsourced API documents in Stack Overflow and found that a larger proportion of existing documents addresses a smaller portion of topics. They leveraged this inequality and proposed a method that projects a crowdsourced documentation into a concept domain and recovers traceability links based on reusability of the concept domain. They claim that their method improves documentation coverage by 400%, by reusing existing documentations that are related to popular (i.e., having high views) concepts or classes on Stack Overflow for unpopular (i.e., having low number of views) concepts or classes.

Barzilay et al. in [4], explored in their work, the design and characteristics of Stack Overflow and developed a code search tool, *Example Overflow*, on top of Stack Overflow. Their code search tool is capable of extracting high quality code examples from Stack Overflow. As part of their empirical analysis, they studied the type of questions posted on Stack Overflow and to what extent these questions can be answered by their approach. Subramanian et al. [29] proposed a method for linking code examples posted on Stack Overflow to API documentation. Based on the proposed method, they implemented a tool, *Baker*, that links code snippets to Java classes and methods or JavaScript functions, with an observed precision of 97%.

Many exploratory studies have been conducted to analyze the characteristics of textual crowd-based documents (e.g., [6], [64], [65]). Parnin and Treude [6] investigate in their work the possibility of crowd-based documents replacing traditional software documentation. They measured in their work the extent to which blog posts cover methods of a particular API. They observed that social media posts can cover 87.9% of the API methods and therefore could potentially replace traditional documentation. Nasehi et al. [66] analyzed code examples of Stack Overflow that are voted by users as being good code examples to identify the characteristics that, if applied, can improve the development and evolution of API documentation.

Campbell et al. [64] combined Stack Overflow questions and PHP and Python projects' documentation and used LDA and found topics in Stack Overflow that did not overlap the topics on projects' documentation. Their results also show that many topics that are covered on Stack Overflow but not by traditional project documentation, are related to external project documentation or tutorials. Pham et al. [65] investigate the possibility of extracting a common testing culture to tackle challenges that social coding sites introduce to testing behavior. They conducted a survey among GitHub users to identify the challenges, the impact of these challenges on testing practices and based on the survey results they suggest strategies that, if applied by software developers and managers, can positively affect the testing behavior in their projects.

While this existing research analyses how developers can use textual content extracted from crowd-based documents as a source of documentation, our work focuses on the extraction and integration of crowd-based *multimedia content* with source code.

4.2 Text Retrieval-based Feature Location

Several feature location techniques have been proposed in the literature, with a detailed overview of these techniques being presented elsewhere in [51]. In what follows, we discuss the work that is most closely related to our approach.

Marcus et al. [67] leveraged Latent Semantic Indexing (LSI) [68] to find semantic similarities between a query that is either automatically generated or provided by the user to locate source code features. Van der Spek et al. [69] also used LSI to locate features in source code. They applied different preprocessing and text normalization methods, such as stemming, stop words removal, common terms weighting to their data sets and then evaluated effect of each processing step on their feature location approach.

To improve feature location results, Bassett et al. [35] proposed a new term weighting technique that uses the structural information in the source code (i.e., function names and method calls extracted from call graphs). The new term weighting technique is an LDA-based approach that is used to overcome the limitations of applying tf-idf term weighting in source code-related text since tf-idf is designed for unstructured documents written in natural language. Their evaluation shows that the LDA-based approach delivers more accurate results. Eddy et al. [70] expanded the work by Bassett et al. [35] by exploring a broader range of criteria in the source code (e.g., leading comments, method names, parameters, body comments, and local variables) and weighted them according to their position in a method. Lukins et al. [71] proposed an LDA-based bug localisation technique using queries that are extracted and formulated out of bug reports. They evaluated their approach by calculating the percentage of bug queries whose first relevant result is in the top 10 or top 1,000 results.

Common to these existing approaches is that they use LDA or LSI to locate software artifacts through queries that are either provided by a human or automatically generated from information found in bug reports or source code data sets. In contrast, our approach applies LDA to extract high-level information about features extracted from screencasts and then links this information to the implementation of these features. Furthermore, to improve our linking results, we apply term weighting to modify the term frequencies in screencasts using corpus-wide term frequencies.

4.3 Linking and Analyzing Software Engineering Screencasts

The first study on using crowd-based screencasts to share and document developer knowledge was conducted by MacLeod et al. [2], who investigated why developers create screencasts. They also discuss the benefits and challenges of this type of knowledge sharing by analyzing 20 tutorial screencasts and interviewing 10 developers/YouTubers. They found that by using screencasts developers demonstrate and share information related to how to customize a program, the challenges they encountered and their development experiences, solutions to problems, how to apply design patterns, and their programming language knowledge. They also observed that developers are creating these screencasts to promote themselves and gain reputation by helping others. MacLeod et al. [9] performed a study of Ruby on Rails screencasts comparing screencasts published through free platforms (i.e., YouTube) with screencasts specifically designed for a specialized paid platform (i.e., RailsCasts). As part of this study, the authors extract guidelines for screencast creators on how to produce clear and understandable videos.

Other research studied the usage of crowd-based tutorial screencasts in the software engineering domain. Amongst the earliest work in this area is that of Bao et al. [15], [72], who developed a video scraping tool, *scvRipper*, to automatically extract developers' behavior from screencasts. They extract actions of a developer by employing key point-based template matching based on visual cues in an image (e.g., icons that appear in a window). Key points in an image are what stand out in an image and they can be identified even after applying transformation such as resizing or rotating the image.

Although *scvRipper* is not sensitive to screen resolutions and window color schema, the requirement for applications in screencasts to use the same layout and window structures reduces the generalizability of the approach. Bao et al. [45] proposed a method to track user activities and developed a tool called *ActivitySpace* to support inter-application information needs of software developers. The tool reduces the effort required by developers for locating documents and recalling their history activities in daily work.

In [16], Bao et al. also introduced a tool, *VT-Revolution*, that captures the workflows in programming tutorials and enables timeline-based browsing of tutorials as well as accessing the API documentation of a selected code element. In their approach, they record the low-level Human-Computer Interaction (HCI) data used by tutorial creators while working with screencast authoring tools. Khandwala et al.'s *Codemotion* [13] used a computer vision algorithm to extract source code and dynamic code edit intervals from tutorial screencasts. The tool uses a video player UI to enable code search and navigation. In our previous work [18], we extract documentation from the speech component of tutorial screencasts that describe how to use the features of a GUI application. A case study evaluating how this approach can be used to extract use-case scenarios from a total of 25 WordPress screencasts of 5 scenarios showed that the approach extracts use-case scenarios from screencasts with a median precision and recall of 83.33% and 100%, respectively.

Other work [2], [3], [9], [12] addresses the need for designing concise video tutorials that have less noise and can precisely describe what viewers need [12]. Ponzanelli et al. [3], [12] developed an approach to extract relevant fragments of programming tutorial videos and link them to relevant

StackOverflow discussions by mining the (captioned) speech and GUI content of the video tutorials. Yadid et al. [73] developed an approach to extract code from programming video tutorials to enable deep indexing of these videos. Their approach consolidates code across multiple image frames of the videos and uses statistical language models to make corrections on the extracted code. Another work by Ott et al. [74] presented an approach that uses deep learning, and more specifically convolutional neural networks and autoencoders, to identify source code examples in image frames of a large data set of videos.

Escobar-Avila et al. [75] and Parra et al. [76] presented text retrieval-based tagging approaches to help users to identify whether or not the content of a video tutorial might be relevant to the needs of a user. Poche et al. [77] classify tutorial video comments using Support Vector Machines (SVM), to summarize the comments for content creators. Ellmann et al. [78] used a frame similarity approach (i.e., cosine similarity and LSI) to identify and distinguish development screencasts from other types of videos on YouTube and link them to their relevant API documentation using only the audio transcript of the videos.

Common to this related research is that they use programming tutorial videos as input and try to extract source code or programming documentation from them and link these code artifacts to corresponding source code artifacts. The approaches often rely on a combination of natural language processing (NLP), image processing, and machine/deep learning techniques. In contrast to this existing work, our goal is to analyze and link screencasts that do not show source code and instead demonstrate or provide walkthroughs, using features/options which are typically part of the GUI of the actual application. In our approach we link high-level GUI information of an application that is shown in a screencast to the corresponding source code artifacts of the application using LDA.

4.4 Discussion

The research works presented in Sections 4.1 and 4.3 mostly focused on introducing crowd-documentation as a supplementary source of information and a source that can be mined to extract new useful information (e.g., testing practices). These works paved the way for exploring a wider range of possibilities in extracting information from crowd-based documentation and analyzing more challenging documents such as videos. This motivated researchers (Section 4.3) to work on how to improve the quality of crowd-based tutorial videos and propose approaches that accurately index, tag, or leverage such documents. However, due to the level of experience and/or generation gap, multimedia documentation is not a type of reference that every software stakeholder would be interested in using. None of the reviewed works in Sections 4.1 and 4.3 are motivated to close this gap by working to provide a semantic integration of screencast content with other software artifacts through which all stakeholders would have access to their preferable type of documentation (e.g., textual use-case scenarios that are extracted from tutorial screencasts) as well as their relevant software artifacts in the project (e.g., source code, bug reports).

Among the reviewed related works in Section 4.3, a few ones attempt to link screencasts to software artifacts that are relevant to the feature being demonstrated in the screencast, because this is a challenge that is different from using programming tutorial screencasts and requires using more advanced IR (Information Retrieval) and NLP techniques. The challenges mainly arise from the fact that such screencasts contain high-level and less technical information about a certain functionality of an application while their relevant software artifacts (e.g., source code) contain low-level and more technical information.

The reviewed works in Section 4.2 show how other researchers tackled the problem of locating software artifacts that are relevant to a query by using LDA and LSI approaches. Our proposed research is the only work that uses the text extracted from **screencasts containing high-level information (i.e., GUI-based) and applies LDA to it** to extract information (e.g., use-case scenarios) from them or link them to other artifacts (e.g., source code that may even contain low-level implementation of the corresponding features).

4.5 Chapter Summary

In this chapter, we reviewed the most closely related research to this thesis and discussed and identified the current research gaps such as providing access to a type of documentation that is preferable by different software engineers and linking screencasts containing high-level information (i.e., GUI-based) to their relevant technical or source code artifacts. In the next chapter, we present our user survey analysis results. This analysis provides the basis for our methodology of mining and linking screencasts.

5 Adoption of Crowd-based Software Engineering Tutorial Screencasts

5.1 Introduction

Research [79] has shown that using tutorial videos as a learning aid for traditional lectures can significantly improve students' performance in education. Also, when compared to written text, screencasts can provide a clearer explanation of a subject matter while also being accessible through different devices [80]. The latter work also showed that visual learners and students benefit from screencasts by being able to learn at their own pace, whenever and wherever they want [80].

In software engineering, tutorial videos are used as a medium to share information for different purposes (e.g., learning, technological trends, job training, how-to guides) [1]. In this chapter, we report on a survey[10] that we conducted with software engineers to further investigate the relevance of using tutorial screencasts as an information source in software engineering.

As discussed in Chapter 2 and our motivating example, we aim to gain insights on how a user's background, more specifically, age range (i.e., Gen X, Gen Y, and Gen Z) and level of professional experience influence the use of certain documentation media. We focus on tutorial screencasts that are designed for different purposes (e.g., programming, getting familiar with GUI features, learning new concepts, etc.) to identify their important components (i.e., information sources) and provide insights on their usage (e.g., based on the task to be performed), advantages, and disadvantages from a practitioner's perspective.

We describe our approach of analyzing the responses and present the results of our analysis. Based on the results, we discuss how software engineers benefit from screencasts as well as challenges they face in using them as project documentation. Finally, we discuss potential avenues on how to improve the usefulness of screencasts and to speed up their adoption in the software engineering domain.

[10] The analysis of the survey that is presented in this chapter is submitted to Information and Software Technology (IST) Journal.

By analyzing the survey responses, we try to answer the following research questions:

- **RQ1**. What type of documentation is preferred by software engineers based on their level of experience or age?
- **RQ2**. How often and why do software engineers watch how-to tutorial videos?
- **RQ3**. What are the important components of a tutorial video according to its users?
- **RQ4**. How are tutorial videos used as a software artifact?

5.2 Survey

In this section we describe the design and setup of our survey questionnaire and provide some insights into the demographics of our survey respondents, data preparation and, how we processed open-ended questions.

5.2.1 Survey Setup

We conducted a survey (see Appendix) consisting of 33 open questions including 29 multiple choice and five- and six-level Likert scale[11] questions (e.g., Always, Often, Sometimes, Rarely, Never) and 4 open-ended questions. We designed the questionnaire such that it can be completed in approximately 15 minutes. We used Google forms, which is a widely used platform to create free online surveys, to design and create the questionnaire. The survey was also approved by the Concordia Research Ethics committee. We distributed the link to the survey[12] on LinkedIn[13], Facebook[14], Twitter[15] and invited software engineers and anyone with experience or expertise in programming to participate in the survey. We also distributed the link to software engineering researchers in the MCIS[16] and ASEG[17] labs, and a learning platform[18] that is accessible to both graduate and undergraduate students enrolled in a software engineering course at Concordia University[19]. To avoid bias in the responses, we did not share our research hypothesis with survey

[11] https://www.surveymonkey.com/mp/likert-scale/
[12] https://forms.gle/36rDPAWtRuLrtT1u9
[13] https://www.linkedin.com/
[14] https://www.facebook.com/
[15] https://twitter.com/
[16] http://mcis.polymtl.ca/
[17] https://sites.google.com/view/juergenrilling/home
[18] https://moodle.concordia.ca/moodle/
[19] https://www.concordia.ca/

participants. We assured the participants that we intend to publish the aggregated results of the research and that it will not be possible to identify individual participants in the published results.

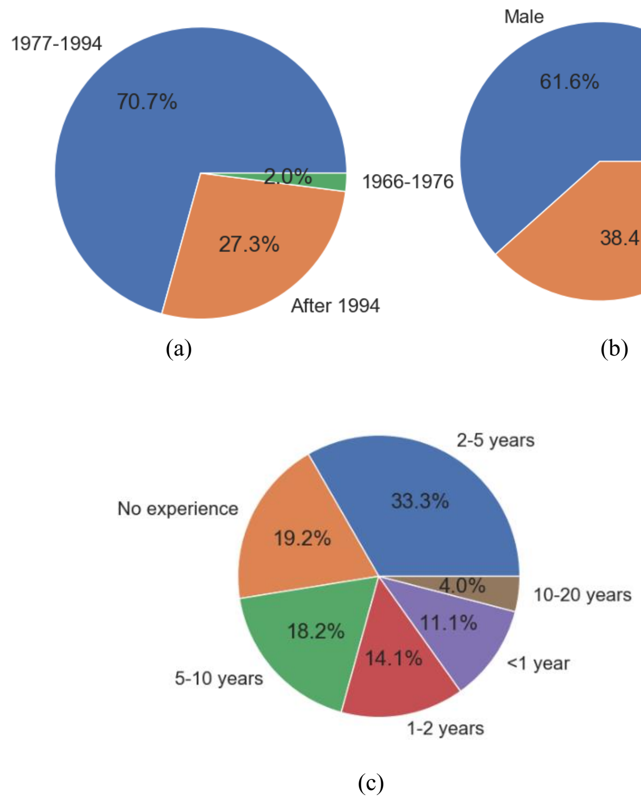


Figure 5-1. Age range (a), gender (b), and software engineering-related experience (c) of the survey participants

The questionnaire is designed in four parts, which we briefly describe here. The first part is used to gather background information of the survey participants such as their gender, birth year, occupation, years of professional experience as a software engineer, programming languages that they use and, their open source experience.

The second part of the questionnaire addresses questions such as: 1) what type of documentation (digital, non-digital, and social media) is preferred by the participants, to get help with their software engineering tasks (e.g., using an application feature, using an API, etc.) or learn something (e.g., a programming language, how to use a tool, etc.), 2) the importance of different social media channels (e.g., GitHub, Stack Overflow, YouTube, etc.) in completing their software engineering tasks and, 3) the type of media and the format that they prefer to use to learn new concepts or use as documentation. The answers to these questions are not mutually exclusive and multiple answers are possible. The feedback collected from this part of the survey will be used to find out what source of documentation is preferable by software engineers, with different characteristics, to get help with their software engineering tasks.

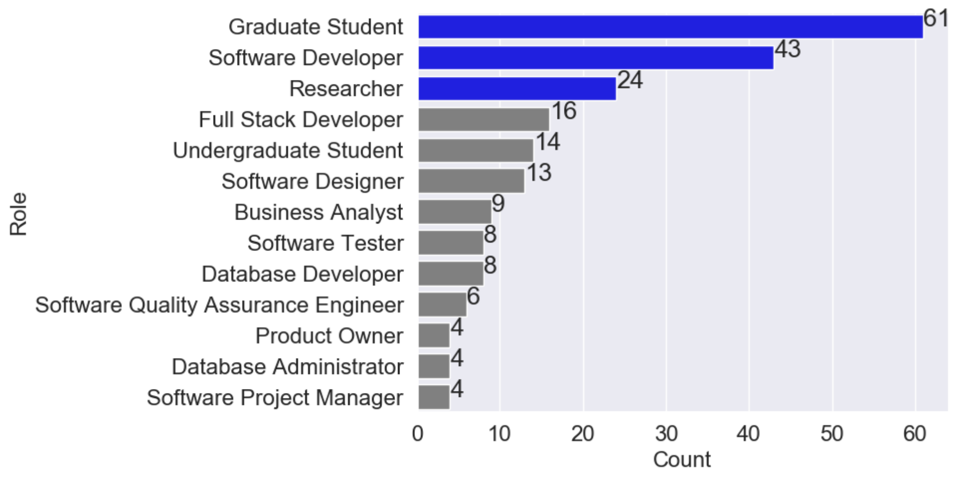


Figure 5-2. Role of the participants. Options that were selected by more than 20 participants are colored in 'blue'.

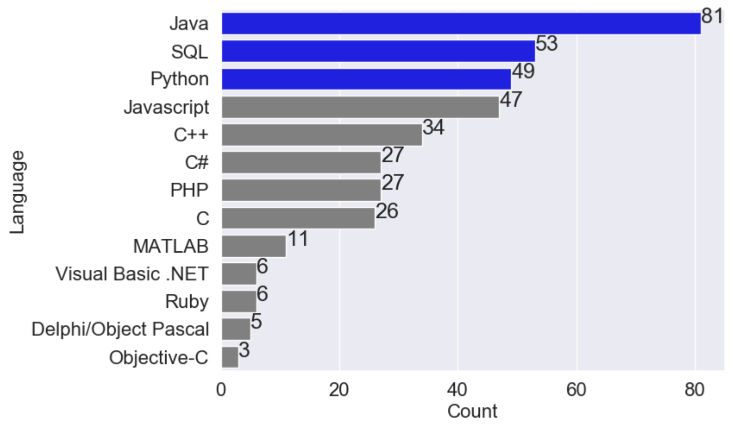


Figure 5-3. Programming languages that participants use for development or maintenance tasks. Top 3 languages are colored in '"blue"'.

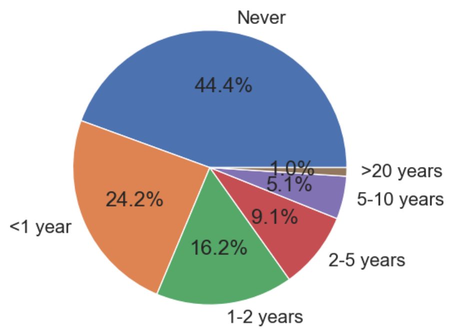


Figure 5-4. Years of open source contribution.

In the third part of the questionnaire, we want to gain insights on how often and why software engineers watch how-to tutorial screencasts. We included questions asking why participants watch software engineering-related tutorial screencasts, how often they view tutorial screencasts as a learning resource and how often they find such tutorials effective and if they find their content useful. We also included a set of questions to help us identify what users consider the most important aspects of a tutorial screencast, including their level of satisfaction with the audio and video quality of screencasts, and what criteria they use when selecting a tutorial video (e.g., author, content, speech, HD quality, number of views, number of likes, title).

In the third part, we provided sample scenarios that feature a software engineer who wants to learn how to perform a specific task related to both a high-level use-case scenario of an application and a more technical scenario related to an implementation issue of this software application. We then asked the survey participants what type of documentation (Stack Overflow, the official online documentation, a tutorial screencast, or all of them) they would use to perform the given task. In addition, to better understand what our participants consider as advantages and disadvantages of using screencasts compared to text-based documentation, participants were asked to complete open-ended questions.

Finally, in the fourth part of the questionnaire, we asked participants how they would use how-to tutorial screencasts as a software artifact, when contributing to an open source project they are unfamiliar with. With this assumption in mind, we asked them how frequently they watch a screencast related to a software project whose source code they want to customize or how often they try to locate the source code artifacts of a feature that is being demonstrated in a screencast. We also asked them to specify the type of screencasts (e.g., screencasts relevant to online textual documentation, screencasts relevant to security issues, screencasts relevant to Stack Overflow question and answers, etc.) they find to be most relevant.

5.2.2 Demographics of the Participants

We received 99 responses from participants with different software engineering backgrounds. Figure 5-1 to Figure 5-4 provide a general overview of the demographics of the survey participants. Figure 5-1b shows that the majority of the survey respondents are male (61.6%) compared to 38.4% being female.

Figure 5-1a shows the age distribution of the survey participants, with approx. 70% of the participants being born between 1977 and 1994 (therefore considered to be Gen Y), 27.3% being born after 1994 (Gen Z) and only 2.0% being born between 1966 and 1976 (Gen X). Most of the survey participants (61.6%) are graduate students or undergraduate students (24.2%) - see Figure 2, with 19.2% of the survey participants having no prior professional experience in software engineering (Figure 5-1c).

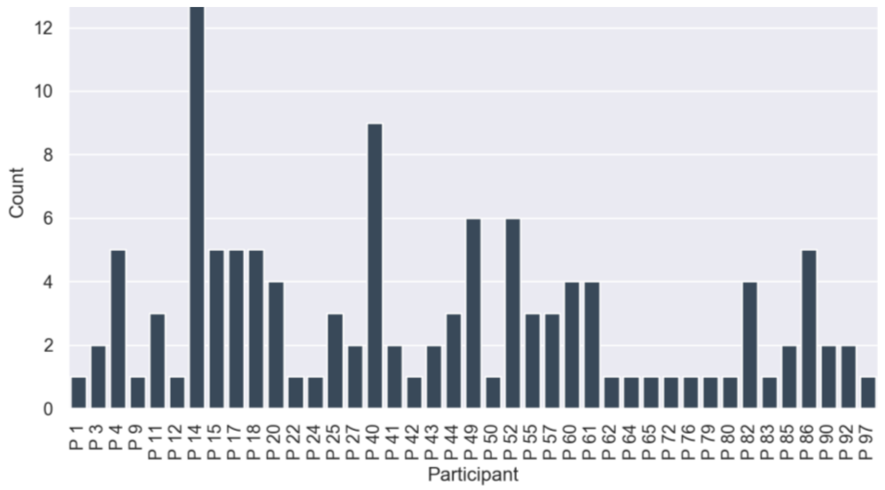


Figure 5-5. Frequency of "Not sure" options selected by individual survey participants.

Figure 5-3 provides an overview of the programming languages most commonly used by our participants, with Java (81.8%), SQL (53.5%), and Python (49.5%) being the top three languages. Among the participants, 44.4% have never contributed to any open source project and 24.2% have less than a year of open source development experience (Figure 5-4).

5.2.3 Data Preparation

To answer our research questions, we preprocessed our collected data. In case of our six-level Likert scale responses, we provided the option of "Not sure" to the respondents so that when they do not have a clear answer for a question, they can select this option. During preprocessing, for each question, we removed the "Not sure" responses. For instance, for a question that asks about the level of preference of a person using a certain type of documentation for a particular type of task, we provided Likert scale levels such as: "Highly preferable", "Moderately preferable", "Somewhat preferable", "Not very preferable", "Not preferable at all", "Not sure". In this case we removed all respondents that selected the "Not sure" from this specific question, since it did not provide any additional insights. Our survey (see Appendix) contains 7 questions with "Not sure" answers (questions 8, 10, 11, 20, 21, 23, 27), for a total of 35 possible "Not sure" answers. Figure 5-5 shows the frequency for which the "Not sure" option was chosen by our participants. The figure includes only participants who chose the option "Not sure" at least once.

We also merged the levels of professional experience (Figure 5-1c). Since most of our participants are graduate students and we also have undergraduate students in our respondents (Figure 5-2) we observed that most of our participants have less than 5 years of professional experience in software engineering. Therefore, based on the observed distribution of the data (Figure 5-1c), we merged "No experience" and "<1 year" into "<1 year" and also we merged "5-10 years" and "10-20 years" into ">5 years".

In case of our participants' age groups, we only had two respondents who were born between "1966-1976" (Figure 5-1a). Given this small number, we decided to merge their responses with those whose birth year is in the range "1977-1994", which resulted in a dataset with 72.7% of the participants being Gen Y/X and 27.3% being Gen Z.

5.2.4 Analyzing Open-ended Questions

In our questionnaire, we asked three open-ended questions to let the participants share their responses in free form text. We analyzed the answers to these questions following a grounded theory approach [81] to pinpoint repeated answers and ideas in written responses. With the help of a Ph.D. colleague the following steps are taken to code the answers:

1. One of the raters identified the main categories in the respondents' answers and shared the topics with the other rater.
2. Both raters independently reviewed the answers and tagged them for important codes in each category identified in the previous step.
3. The raters shared with each other the (potentially new) codes that they identified in the second step and had a discussion to agree on their identified codes and built a codebook.
4. Then, both raters tagged the answers once again using the codebook that was created in step 3.
5. Finally, we calculated inter-rater reliability using Cohen's kappa to measure the degree of agreement among the raters (Table 5-1).

Table 5-1. Interpretation of Kappa statistics.

Kappa Statistic	Agreement
< 0	Less than chance
0.01 – 0.20	Slight
0.021 – 0.40	Fair
0.41 – 0.60	Moderate
0.61 – 0.80	Substantial
0.81 – 0.99	Almost perfect

5.3 Study Results

In this section, we analyze our survey responses to answer the research questions introduced in Section 5.1.

RQ1. What type of documentation is preferred by software engineers of different level of experience or age?

Figure 5-6a provides an overview of the respondents based on their years of professional experience and their preferred source of developer knowledge (regardless of its textual or multimedia format) they use to get help for their software engineering tasks. Participants could select between "Digital" (e.g., Slashdot, Sourceforge, Visual Studio documentation, Eclipse documentation, mailing lists, emails), "Non-digital" (e.g., telephone, face2face, project workbook, documents, books) or, "Social media" (e.g., blogs, Twitter, LinkedIn, YouTube, Vimeo, Stack Overflow, Slack, Facebook, GitHub). Among the responses for "Non-digital" resources, two persons and for the "Social media" preference, one person responded, "Not sure". These responses were removed from our analysis since they are not informative. Our survey results show that **"Non-digital" knowledge resources are the least preferred** among participants. The results also show that **social media resources gain in popularity among the more experienced** software engineers.

We tested the statistical significance of our observations between the level of experience and using "Digital", "Non-digital" or, "Social media" resources using Chi-squared statistical tests with alpha=0.05. We found that respondents with different level of professional experience have different preference in using "Digital" knowledge resources, since this observation is statistically significant, with a p-value = 0.02. However, the result of the same test for the "Non-digital" and "Social media" knowledge resources were not statistically significant.

Classifying the respondents based on their age and their preference of using “Digital”, “Non-Digital” and, “Social Media” as their information resource shows that **as participants get older, they rely more frequently on “Social Media” sources** (Figure 5-6b). However, the results are not statistically significant (using Chi-squared statistical tests with alpha=0.05).

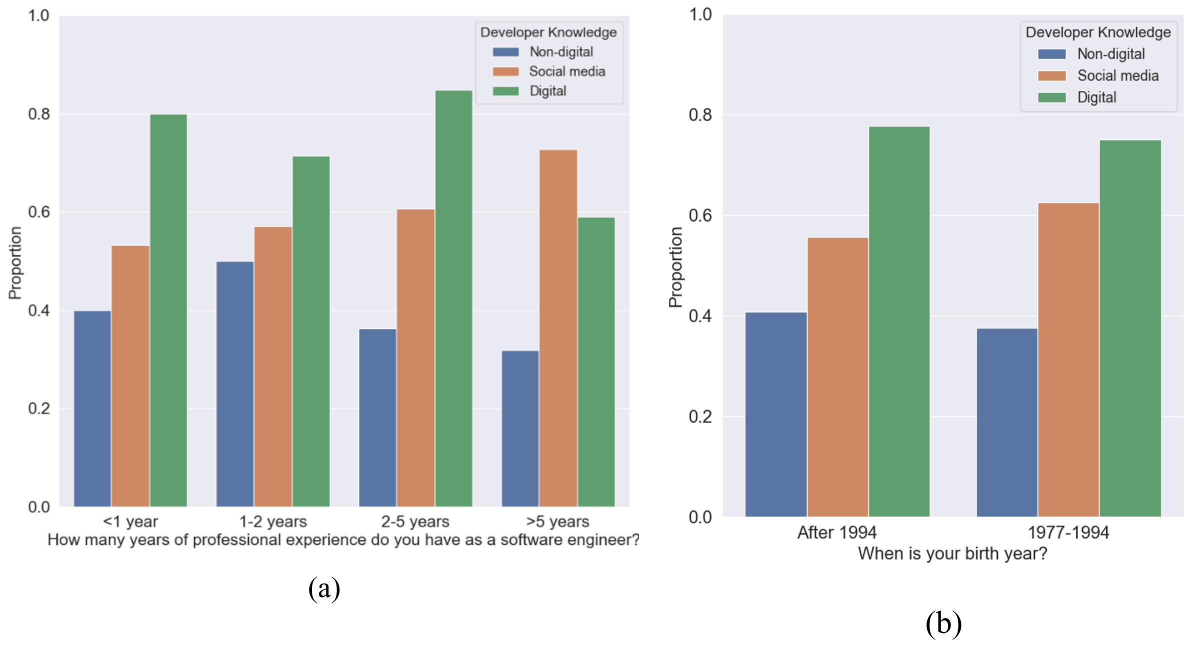


Figure 5-6. Preference of developer knowledge to get help from in software engineering tasks based on: professional experience (a), and age group (b).

Based on the received responses (Figure 5-7), **“Stack Overflow”, “GitHub”, and “YouTube” are among the top three social media resources** that our participants consider to be important “to complete their software engineering-related tasks”. These resources had originally three, two, and one “Not sure” responses respectively that were removed prior to our analysis. The analysis of the responses (Figure 5-7) shows that, as developers become more experienced, they find “Stack Overflow” to be more important than other media types and the use of “YouTube” significantly drops among more experienced professionals with p-value = 0.025 for a Chi-squared statistical test applied between levels of experience and media types (YouTube, Stack Overflow or, GitHub). Also, more experienced (>2 years) software engineers rely less frequently on “GitHub” compared to the less experienced professionals (<2 years). The same statistical test of participants with different levels of experience having different levels of preference in using Stack Overflow and GitHub did not show any statistical significance.

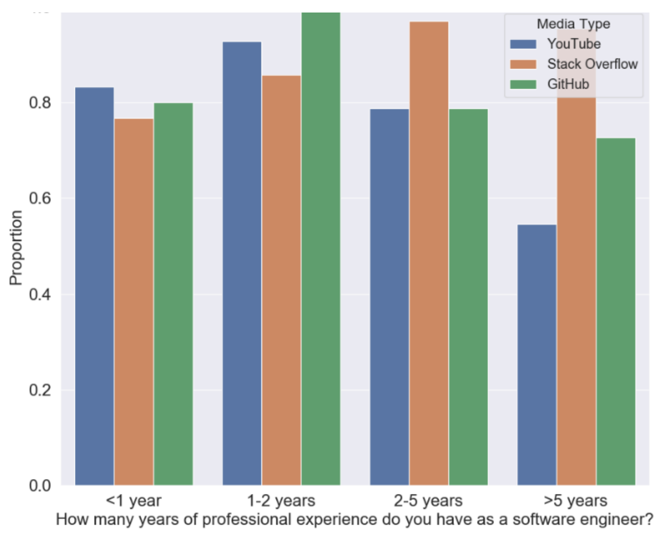


Figure 5-7. Importance of different media types to complete software engineering tasks based on professional experience.

We further asked our participants to rank their preference in using “Books”, “Blog posts”, “Images”, “Online written documentation”, “Podcasts”, “Question and answer sites”, and “Videos” for learning new skills/concepts (i.e., “Videos” in general, meaning that they can be related to programming or non-programming skills/concepts). The survey results show that **“Videos” are the most preferred (i.e., “Highly preferable” or “Moderately preferable”) and “Podcasts” the least preferred (i.e., “Not very preferable” or “Not preferable at all”)** media types. “Question and answer sites” and “Online written documentation” are also among the common sources that respondents chose to learn new skills/concepts.

We further analyzed the responses to see whether the number of years of professional experience has an impact on the preferred media for learning new skills/concepts. Figure 5-8 shows that **as participants gain work experience, in addition to “Videos” they more progressively rely on “Question and answer sites”** as their primary source to learn new skills/concepts. However, the results are not statistically significant. This is while the proportion of the participants who prefer “Videos” as a learning source always remains above 80% in all categories (i.e., levels of experience).

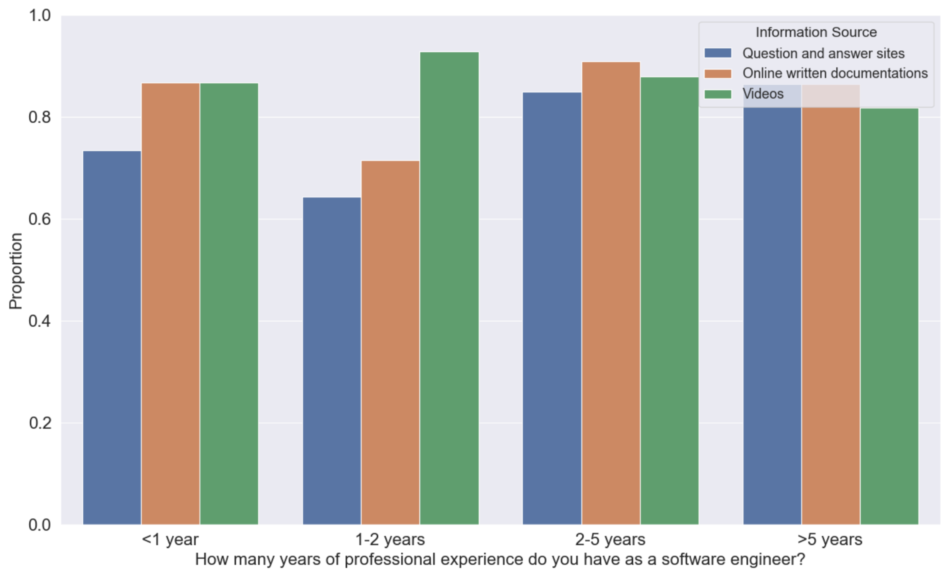


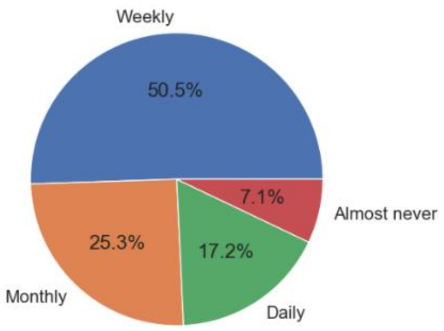
Figure 5-8. Preference of different information sources to learn new skills/concepts based on professional experience.

Conclusion: While more experienced software engineers prefer “Question and answer sites” such as “Stack Overflow” to get help or improve their current skill set, less experienced (<2 years) software engineers mostly prefer “YouTube” or “Videos” as their information sources to complete their tasks or learn new skills.

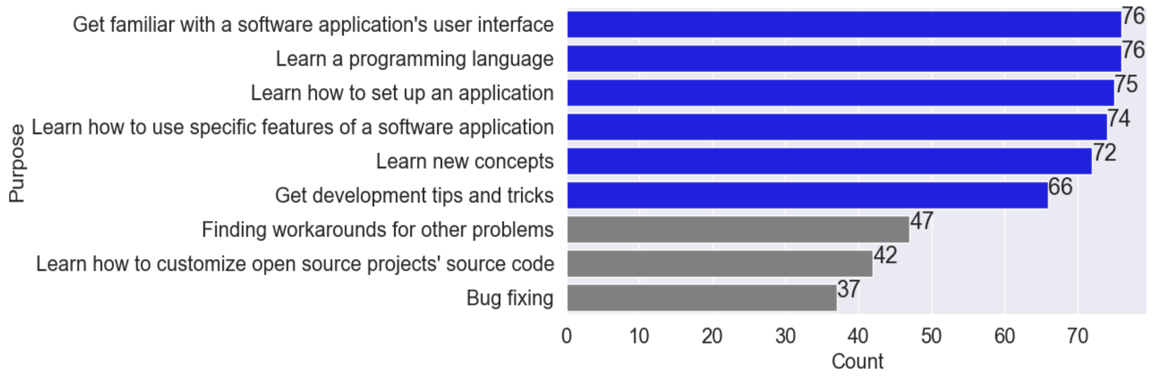
RQ2. How often and why do software engineers watch how-to tutorial screencasts?

Our survey also shows that **half of the participants (50.5%) watch tutorial videos as a learning resource on a regularly (weekly) basis** (Figure 5-9a).

A more detailed analysis shows that **software engineers watch most commonly screencasts to “Learn a programming language” (i.e., programming screencasts) or “Get familiar with a software application user interface” (i.e., non-programming screencasts)** (Figure 5-9b), while watching videos for “Bug fixing”, “Learn how to customize open source projects’ source code” (which are programming-related purposes of watching screencasts), and “Finding workarounds for other problems” are less common reasons among all participants (i.e., selected by less than 50% of the participants).



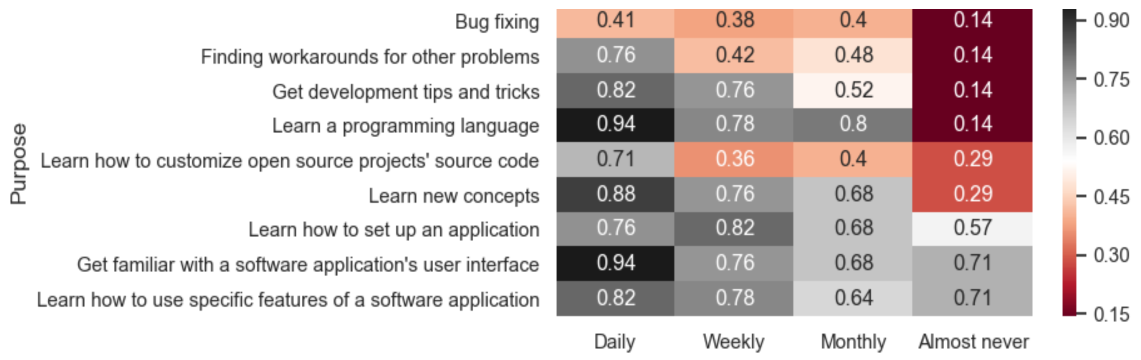
(a)



(b)

Figure 5-9. Number of participants watching software engineering-related tutorial screencasts (a) and their motivation for watching them. Options that were selected by more than 50 percent of the participants are colored in 'blue' (b).

Among these topics for watching tutorial screencasts, "Bug fixing", "Finding workarounds for other problems", "Get development tips and tricks", "Learn a programming language", and "Learn how to customize open source projects' source code" are more technical and programming-related topics, while "Learn new concepts", "Learn how to set up an application", "Get familiar with a software application user interface", and "Learn how to use specific features of a software application" are rather high-level and less technical topics.



Purpose	Daily	Weekly	Monthly	Almost never
Bug fixing	0.41	0.38	0.4	0.14
Finding workarounds for other problems	0.76	0.42	0.48	0.14
Get development tips and tricks	0.82	0.76	0.52	0.14
Learn a programming language	0.94	0.78	0.8	0.14
Learn how to customize open source projects' source code	0.71	0.36	0.4	0.29
Learn new concepts	0.88	0.76	0.68	0.29
Learn how to set up an application	0.76	0.82	0.68	0.57
Get familiar with a software application's user interface	0.94	0.76	0.68	0.71
Learn how to use specific features of a software application	0.82	0.78	0.64	0.71

Figure 5-10. Proportion of participants with different frequency of watching tutorial screencasts based on their motivation for watching screencasts.

We further analyzed our **participants' motivation for watching screencasts based on how frequently they watch tutorial screencasts** (Figure 5-10), and we analyzed if the work context of a person affects the tutorial watching behavior (Figure 5-11). To gain a better insight into the results we partially ordered the purposes of watching tutorial screencasts based on the proportions of participants that fall in each category in Figure 5-10 and Figure 5-11.

We then analyzed the responses where the participants indicated that they **"Almost never"** watch tutorial videos as a learning resource (Figure 5-10). We noticed that **this group of users, if they watch tutorial videos, watch them to "Get familiar with a software application's user interface" (71%) or to "Learn how to use specific features of a software application" (71%).** Among the **survey participants who indicated that they watch tutorial videos on a daily basis (17.2%), they watch them mostly to "Learn a programming language" (94%), to "Get familiar with a software application's user interface" (94%), or to "Learn new concepts" (88%).**

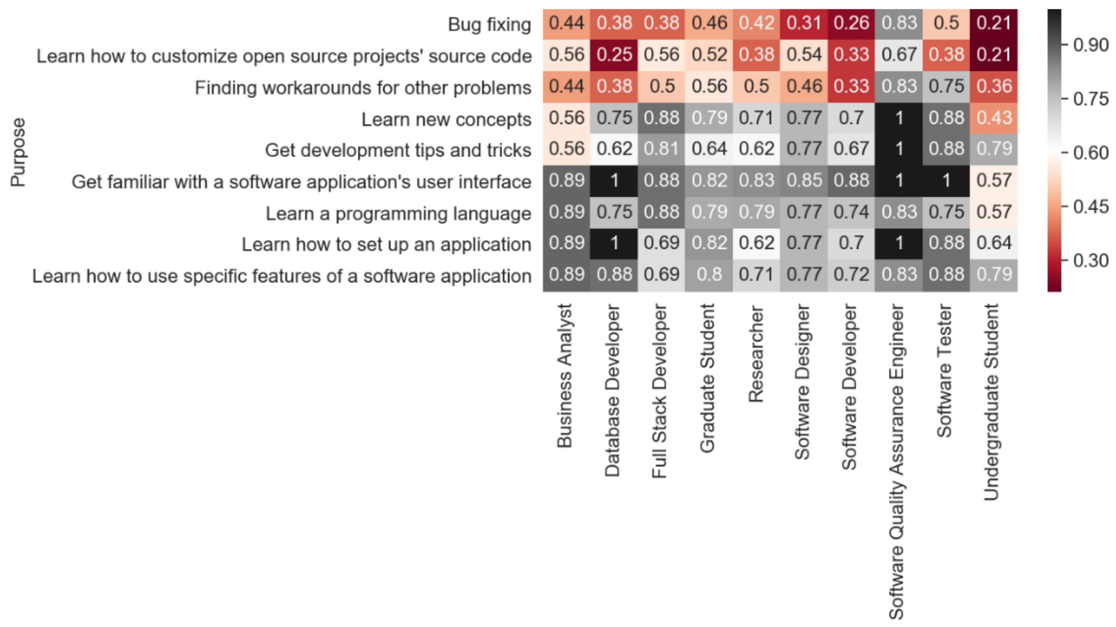
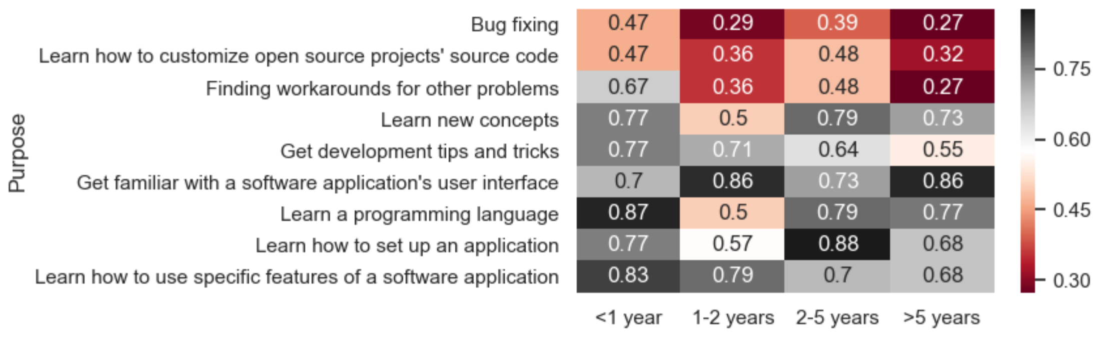


Figure 5-11. Proportion of participants with different role/occupation based on their motivation for watching screencasts

	Business Analyst	Database Developer	Full Stack Developer	Graduate Student	Researcher	Software Designer	Software Developer	Software Quality Assurance Engineer	Software Tester	Undergraduate Student
Bug fixing	0.44	0.38	0.38	0.46	0.42	0.31	0.26	0.83	0.5	0.21
Learn how to customize open source projects' source code	0.56	0.25	0.56	0.52	0.38	0.54	0.33	0.67	0.38	0.21
Finding workarounds for other problems	0.44	0.38	0.5	0.56	0.5	0.46	0.33	0.83	0.75	0.36
Learn new concepts	0.56	0.75	0.88	0.79	0.71	0.77	0.7	1	0.88	0.43
Get development tips and tricks	0.56	0.62	0.81	0.64	0.62	0.77	0.67	1	0.88	0.79
Get familiar with a software application's user interface	0.89	1	0.88	0.82	0.83	0.85	0.88	1	1	0.57
Learn a programming language	0.89	0.75	0.88	0.79	0.79	0.77	0.74	0.83	0.75	0.57
Learn how to set up an application	0.89	1	0.69	0.82	0.62	0.77	0.7	1	0.88	0.64
Learn how to use specific features of a software application	0.89	0.88	0.69	0.8	0.71	0.77	0.72	0.83	0.88	0.79

We further analyzed how the work context of a person affects their tutorial watching behavior (Figure 5-11). Our analysis shows that**, although "Bug fixing" is overall among the least popular reasons for viewing videos, 83% of the "Software Quality Assurance Engineers" and 50% of the "Software Testers" indicated that they specifically watch videos for this reason**. Also, **"Graduate Students" and "Researchers" are more likely to rely on tutorial videos as a learning resource compared to "Undergraduate Students"**. This might be indicative that undergraduate students are often provided with resources that are specifically designed and provided for their courses and assignments, while graduate students and researchers will often have to find their own resources to learn new material.

We also analyzed if a software engineer's experience plays a role why they watch videos (Figure 5-12). **While participants with less than one year of work experience mostly watch screencasts for learning a programming language, more experienced users watch videos to familiarize themselves with an application or to learn how to setup an application**.



Purpose	<1 year	1-2 years	2-5 years	>5 years
Bug fixing	0.47	0.29	0.39	0.27
Learn how to customize open source projects' source code	0.47	0.36	0.48	0.32
Finding workarounds for other problems	0.67	0.36	0.48	0.27
Learn new concepts	0.77	0.5	0.79	0.73
Get development tips and tricks	0.77	0.71	0.64	0.55
Get familiar with a software application's user interface	0.7	0.86	0.73	0.86
Learn a programming language	0.87	0.5	0.79	0.77
Learn how to set up an application	0.77	0.57	0.88	0.68
Learn how to use specific features of a software application	0.83	0.79	0.7	0.68

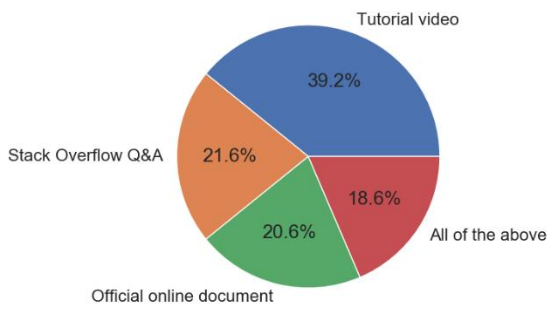
Figure 5-12. Participants' motivation for watching software engineering-related tutorial screencasts based on their years of professional experience.

The **majority of our survey participants find tutorial videos an effective learning tool for software engineering** (i.e., 49.5% "Very effective" and 34.3% "Moderately effective") and the information presented in these videos to be useful (i.e., 32.3% "Very useful", 48.5% "Moderately useful").

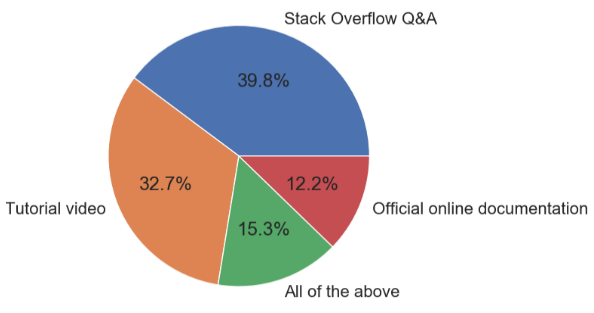
In addition, we analyzed the responses to see whether tutorial screencasts are a popular learning resource for technical and non-technical tasks. As part of our survey, we provided the participants with the following two scenarios using WordPress[20], a well-known content management system: 1. How to add a blog post to WordPress (non-technical) and 2. How to restrict WordPress site access by IP (technical). We then asked the survey participants to indicate which type of documentation they prefer to use for each of the two scenarios.

Figure 5-13 shows that, while the two top choices for both types of tasks are "Tutorial video" and "Stack Overflow", **for the non-technical task, users prefer to watch "Tutorial video" as the most popular learning source (Figure 5-13a), while for the technical task "Stack Overflow" is the first choice (Figure 5-13b)**.

[20] https://wordpress.com



(a)



(b)

Figure 5-13. (a) Type of documentation used to learn how to perform a non-technical task, and (b) a technical task (color coded based on the percentages).

Conclusion: Our survey shows that tutorial screencasts are considered by most users to be useful for comprehending and learning new tasks/concepts. While tutorial screencasts are frequently used by all our survey participants for non-technical software engineering-related purposes, less experienced users and software testers/QA engineers use screencasts also for technical and coding purposes.

RQ3. What are the important information sources of a tutorial video according to its users?

While tutorial videos are the most popular source for learning, **27.3% of our participants indicated that they "Often" are unable to find a tutorial that clearly meets their specific needs** and, 41.4% indicated that they "Sometimes" face this issue (

Figure 5-14).

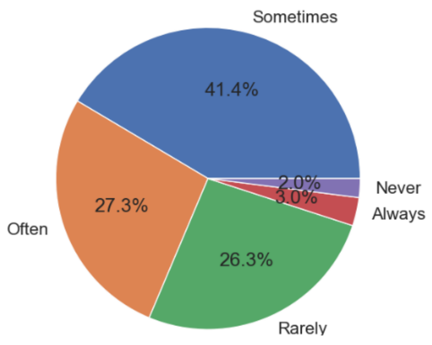
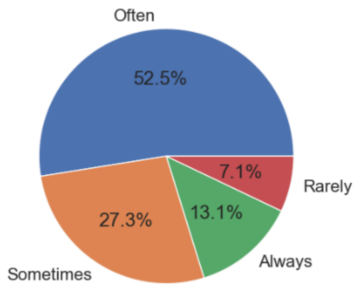
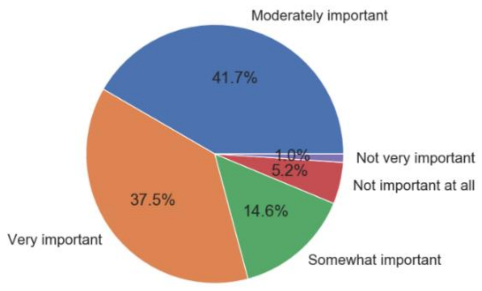


Figure 5-14. The frequency of participants being unable to find videos that clearly describe what they need.

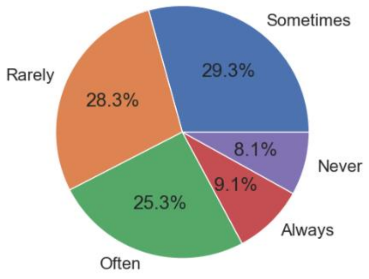
We were further interested in identifying what components (video, audio, closed caption) of a tutorial video are the most important to the viewers. For this, we analyzed responses to survey questions related to the quality of the different information sources associated with a video and how important these sources are when users watch a video. As shown in Figure 5-15a, **most of our respondents (52.5%) indicated that they are "often" satisfied with the audio and visual quality of a tutorial video, and 13.1% indicated that they "always" are satisfied**.



(a) Audio/Visual Quality



(b) Having a Narrator



(c) Turn on Closed Captioning

Figure 5-15. How often are you satisfied with the audio and visual quality of a tutorial video? (a) How important is having a narrator for tutorial videos? (b) How often do you turn on closed captioning (subtitles) while watching a tutorial video? (c) Color coded.

A relevant tutorial video can be located using different criteria such as content (by watching parts of a video) or using meta data, such as title, description, author, number of likes/dislikes, number of views. We therefore asked our survey participants how important each of these information sources (including the length of videos) are when selecting a video. **Most of the respondents (94%) consider the content of a video as an important (i.e., highly important or moderately important) aspect for choosing a tutorial video to watch. The second and third most important criteria are title (84%) and content of the speech transcription (71%) of a tutorial video**. The number of likes (33%), number of views (30%) and, author (30%) of a video are less relevant (i.e., Not very important, Not important at all) for selecting a video.

Furthermore, **providing tutorial videos with narration (and therefore also a speech component) is an important aspect**. The results show that 79% of our respondents consider this a very important or moderately important factor (Figure 5-15b) for selecting a video. Furthermore, **34.4% of our respondents indicated that they often or always turn on the closed captioning while watching a tutorial video** (Figure 5-15c).

Conclusion: Overall, our respondents are mostly using the content (i.e., visual or speech) and title of a tutorial video for selecting which video to watch, while other metadata (i.e., video descriptions, author information and number of likes/dislikes/views) are less frequently used for the selection process.

RQ4. How are tutorial videos used as a software artifact?

The responses (Figure 5-16) show that 31.4% of our participants have at least one year of experience in open source contributions and 24.2% have contributed to open source for less than a year.

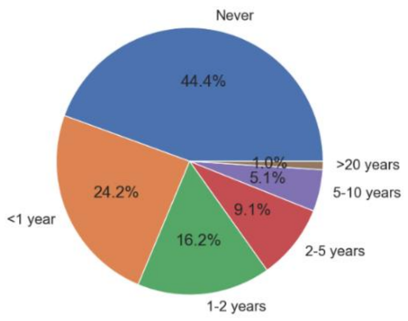
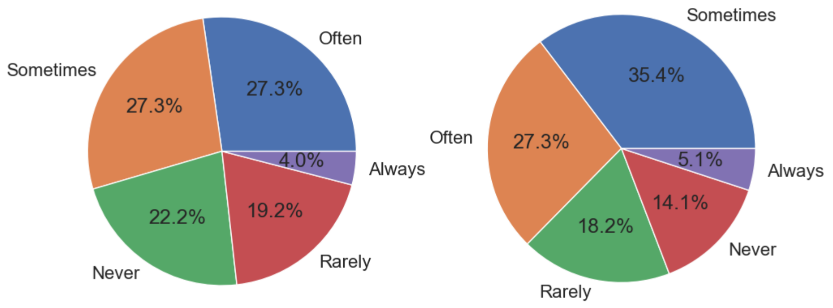


Figure 5-16. How many years the participants have been contributing to open source (in any way).

Given our scenario where a participant is contributing to an open source project without having any prior knowledge about this project and its source code, **27.3% of the participants indicated that in such a situation they *often* try to locate the source code of a feature that is being demonstrated in a video and 4% responded that they *always* try to do so when they contribute to an open source project**. Also, **27.3% of our participants stated that they *often* decide to watch a video of a software project whose source code they want to customize to see GUI/workflow of end user and 5% declared that they *always* try to do so** (Figure 5-17).



(a) (b)

Figure 5-17. How often do you decide to watch a video of a software project whose source code you want to customize? (a) Have you ever tried to locate the source code artifacts of a feature that is being demonstrated in a video? (b) Color coded based on the percent.

We also asked our respondents what technical topics (other than the ones that we explored in the survey, see Figure 5-9) they would find the most interesting. Among our responses (Figure 5-18), "Videos relevant to build configurations" and "Videos relevant to security issues" are the technical topics that are the most interesting to them.

From the results of the last two questions (Figures 5-16 and 5-17) there is an implication that users can benefit from having a bi-directional links between tutorial screencasts (related to: GUI features, build configurations, security issues) and other software artifacts (especially source code). Establishing such traceability links will allow for recommending related documents or artifacts to improve system, subject, or feature comprehension and therefore facilitate performing a change/task.

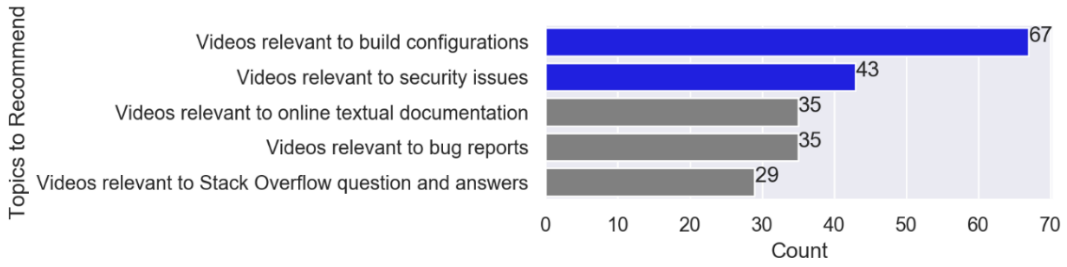


Figure 5-18. The video topics our participants find most interesting/relevant.

We also included an open-ended question for participants to share any suggestion about the role or usefulness of screencasts in software engineering. Using the open coding approach introduced in Section 5.2.4, the following main topics were identified in the answers: (a) content design, (b) accessibility and availability, (c) traceability and, (d) non-relevant answers (i.e., answers not relevant to the question that we asked). We then created a codebook for each of these topics and calculated the inter-rater reliability using Cohen's Kappa for each of the codes (Table 5-2). In what follows, we present our analysis for the codes, where the Kappa statistics show *substantial* or *almost perfect* agreement among the coders (**marked in bold text** in Table 5-2). For the "Accessibility and Availability" category we present the results for the codes whose Cohen's Kappa statistics show *moderate* agreement between the coders (*marked in italic text* in Table 5-2) since this is the highest agreement we have between our coders for this category.

Table 5-2. Topics, codes and, inter-rater reliability of participants' suggestions about the role or usefulness of how-to tutorial videos in software engineering domain.

Topic	Code	Cohen's Kappa
Content Design	Step by Step	0.56
	Audience Expertise	0.45
	To the Point	0.68
	Interesting	0.39
	Interactive	0.32
	Thorough	0.85
	Visual	0.39
	Good Quality	0.85
	Narrator	0.48

Accessibility and Availability	Find Relevant Screencasts	0.37
	Dedicated Platform	0.48
	Create More Screencasts	0.48
Traceability	**Keep Screencasts Up-to-date**	1.0
	Complement Screencasts with Other Documents	0.78
Non-relevant	**Answer is not Relevant to the Question**	0.61

Content Design: Nine of our survey respondents suggested to **design "*short*" tutorial screencasts that provide "*to the point*" information that is "*relevant to the topic*" of the tutorial and does not contain any noise or irrelevant information**. For example, one of the participants stated that "*Tutorial videos should be short and crisp in order to be more understandable".* Another participant mentioned that, *"... One only thing is video should be up to-point and should have proper content as written in the description."*. Three of our participants believe that **information presented in tutorial screencasts that explain a certain task should have no missing/skipped information, with all steps explained in the video to be complete and "*thorough*"** (e.g., "*... Sometimes in videos the steps are skipped*") along with clarifications and explanations (e.g., "*Concept followed by explanation.*"). Other participants also mentioned that **having "*good quality*" audio/visual/content was something that they believe has a positive effect on the usefulness of tutorials in software engineering**.

Accessibility and Availability: This category addresses the issue of creating more videos and making them easier to find. For the following codes our raters had a *moderate* agreement. Two participants suggested to **consider a dedicated platform that contains tutorial screencasts related to software engineering**. For example, one participant mentioned: "*A platform with high quality curated videos would be very useful (provided it's available for free or a minimal fee)*". Two other respondents mentioned that **there should be more tutorials created and made available online** (e.g., "*…it is essential that information in this form should be available as it makes the learning process interesting*").

Traceability: The last major suggestion category is related to creating traceability links between screencasts and other types of software artifacts (e.g., source code, textual documentation, etc.). Two of our participants mentioned that **one issue with using tutorial screencasts is that the version of the software or source code that is being presented in the screencast is often outdated**. For example, one participant stated that "… *the problem is that keeping them up to date* *is hard because of the sequential nature of them*", while another one mentioned that "*It is quite important in the sense that these videos are updated as the relevant software are updated. Else, documentation is more beneficial"*. Screencasts are mostly short and provide visual learning that sometimes lack details and in-depth knowledge about a subject matter. Therefore, eight participants suggested to **complement software engineering-related tutorial screencasts with other types of documentation (online textual documentation, source code, PowerPoint, etc.) or use screencasts as a documentation that is complementary to other documentations**. For instance, one of the respondents' suggestion was: "*It is nice if the video publishers reference to other sources for more details on that particular topic*". Another respondent stated that "*Some people learn faster and better visually; tutorial videos should complement original documentation to provide better user coverage*". Providing well maintained traceability links between screencast and other software artifacts could resolve some of this ambiguity and further improve the interpretation of screencast content.

Conclusion: Our survey not only shows that tutorial screencasts are a popular documentation and learning medium used by many software engineers, but users often follow up on what they have seen in screencasts by trying to trace this video content to the corresponding source code. Our survey participants provided also suggestions on how to improve screencasts in terms of their content design (i.e., to the point, thorough, good quality) and traceability (to keep screencasts up-to-date or complement screencasts with other documents).

5.4 Discussion

In this section, we discuss benefits and challenges of using tutorial screencasts over written documentation. We then discuss our findings and lessons learned from the survey, before drawing some conclusions and propose future directions on how to improve tutorial screencasts in software engineering.

5.4.1 Using Screencasts vs Written Documentation

We asked our participants in two separate questions to share their experience in terms of *benefits/challenges* of using tutorial videos compared to written documentation. Our two raters analyzed the responses to these questions using again the coding approach (described in Section

5.2.4). The two raters identified the major answer categories for each question and created the corresponding codebooks. The categories, their related codes and their inter-rater reliability results (i.e., Kappa's statistics) are shown in Table 3 and Table 4. In what follows, we analyze the responses to these questions and the codes whose Kappa's statistic shows a *substantial* or *almost perfect* agreement between our raters (also marked in bold text in Tables Table 5-3 and Table 5-4). First, we start by analyzing the responses to the question about the *benefits* of using tutorial screencasts over written documentation (Table 5-3).

Content Design: The multimedia format of tutorial screencasts allows video creators to present content in a way that is more "*interesting*" compared to written documentation. Three respondents indicated that, **screencasts are "*more enjoyable*", and "*less boring*"** (meaning easier to follow) **to watch compared to using traditional documentation.**

Table 5-3. The categories, codes and, inter-rater reliability of participants' responses about the benefits of using tutorial screencasts over written documentation.

Category	Code	Cohen's Kappa
Content Design	**Interesting**	0.85
	Interactive	0.31
	Visual	0.32
Accessibility	**Ubiquitous and Easy Access**	0.71
Understandability	Avoid Mistakes	0.49
	Easy Understanding	0.61
	Fast Learning	0.91
	Explanatory	0.64
Negative Opinion	**Had Negative Opinion about Screencasts**	0.85

Ubiquitous and Easy Access: Crowd-based tutorial screencasts are available online in most cases for free. Therefore, software engineers who are seeking help can easily access and use them by streaming them directly from video portals. Two of our participants mentioned that **"*easy access*" and being "*more accessible*" are two of the main benefits of using tutorial screencasts**. Two other participants stated that, **they can "*watch it anywhere and anytime*" and "*access anywhere with Internet facility.*"**

Understandability: Responses that were coded as "*easy understanding*" are the most cited (41% of our survey respondents) benefit of using tutorial screencasts over written documentation. Based on the received responses, **tutorial screencasts are "*much easier to follow*" and "*easier to understand as you can watch the same problem being solved with your eyes*".** Also, we found that people who want to learn a new concept, use screencasts as their preferred learning source, especially for **visual learners** since **"*seeing things in action could result in better understanding of concepts*".** Furthermore, our survey shows that **screencasts help people with remembering and retaining new concepts in memory** since, as mentioned in one sample response, "*learning visually helps us to remember things easily.*"

The visual format of screencasts was stated by 39% of the participants as being **beneficial for "*fast learning*", especially "*for people who are slow-readers*"**. Also, our participants find using screencasts "*time saving*" since "*they convey more comprehensive knowledge in shorter time*" which makes learning "*faster and more efficient*" than when using written documentation.

Another benefit mentioned by five participants for using tutorial screencasts is that **they are easier to understand and more "*explanatory*"** compared to written documentation. One of the main reasons is that the multimedia format of screencasts allows screencast creators to leverage both audio (narrator) and visual (graphical presentations) components to convey their content. For example, participants stated that **"*tutorial videos give a better explanation of the tools/applications being worked with. Also, they show a step by step process on how things are done…*" and that "*videos use more clear words to describe the content than documentation.*"**

Negative Opinions: Although this open question explicitly asked about the *benefits* of using tutorial screencasts compared to written documentation, we also received three coded responses with only negative opinions towards using tutorial screencasts. These **three participants prefer "*textbooks*" over screencasts because, as one of them mentioned, videos lack "*…concrete information. Besides, it is impossible to search videos,*"** and another respondent indicated that **"*while videos focus on specifics, documentation is more detailed and structured. In the long run, reading the documentation has more benefits.*"** In what follows, we elaborate more on these challenges of using screencasts over written documentation resulted from analyzing the four major categories of responses (Table 5-4) identified in our survey.

Reliability: The "*poor quality*" of screencasts is considered by three of our participants as one of the main disadvantages of using screencasts. For example, one respondent mentioned that **videos "*often could not cover the title and lack quality*"**. Around 21% of the survey respondents listed "*lack of details*" as a disadvantage of using screencasts. For example, as **two of the respondents stated "*written can sometimes provide more details over videos*" and "*videos might skip some parts.*"** This may be because video creators try to fit a topic in a short video. For instance, two of the participants indicated that, a) "*The video tutorials are usually an overview of the topic. Detailed tutorials are rarely available. Mostly because the topic is too complex to fit in a video,*" and b) "*… because of the time constraint, videos most often leave out information.*"

In addition, two of the responses are concerned with "*version compatibility*" between the version of the software being presented in a screencast and the version of the application used by the person viewing the screencast. Participants indicated **"*most of the time they are not version based…*" and "*it is very difficult/impossible to update a video, whereas it is easy to update text.*"**

Table 5-4. The categories, codes and, inter-rater reliability of participants' responses about the disadvantages of using tutorial screencasts over written documentation.

Category	Code	Cohen's Kappa
Reliability	Lack of Trust	0.43
	Poor Quality	0.85
	Lack of Details	0.65
	Version Compatibility	0.80
Searchability/Sequential	**Long to Watch**	0.73
	Waste of Time	0.38
	Not Searchable	0.19
	Hard to Follow	0.66
	Not Markable	0.66
Convenience	**Internet**	0.80
	Accent	0.65
	Not Easy to Use	0.66
	Not Copy-pasteable	0.80
Effectiveness	**Not Practicing**	1.0

Searchability: The sequential nature of screencasts limits the ability to directly search for content within the videos. As a result, users prefer short videos over longer videos, since it allows viewers to scan faster through the video to find the relevant content. Also, while locating relevant content, users will often skip non-relevant content, which is easier done for written documentation compared to watching or fast-forwarding through a tutorial screencast that may not even contain the information they are looking for. Around 10% of our participants declared that tutorial **screencasts are "*long to watch,*" meaning that "*they take too long to watch, and they include too much irrelevant information*" and "*time gets wasted*" especially since sometimes "*…they are super time consuming and they don't have a content that actually we want.*"**

Two participants were concerned with tutorial screencasts **being "*hard to follow*" since a) we "*may miss some things in between if we don't listen carefully,*" and b) "*… for a video, you have to go to the pace of the video, but with text you can go at your own pace.*"** Furthermore, two other participants indicated that **screencasts are "*not markable*" and there is "*difficulty in recording points.*"**

Convenience: Based on the received responses, we also noticed that respondents sometimes find it more convenient to obtain required information from written documentation due to different reasons. For example, two participants mentioned **the need for "*Internet*" access/usage as an inconvenience for watching tutorial screencasts since a) "*access to high speed Internet is not available everywhere,*" and b) watching videos requires "*more Internet usage.*"** Also, four of our respondents indicated that **they have difficulty with understanding the "*accent*" of some narrators: "*sometimes the accent of the tutor is hard to understand.*"** One participant mentioned that **screencasts are "*not easier to use than documentation*" and two other participants indicated that screencasts are "*not copy-pastable*" especially for code snippets and commands**.

Effectiveness: Finally, one participant stated that **watching screencasts and "*not practicing*" makes them less effective: "*you might end up in tutorial purgatory, just watching without actually practicing.*"**

In summary, our open-ended questions revealed that content, easy access, easy understanding, fast learning, and being explanatory are among the benefits of tutorial videos over written documentation. Our survey also showed that given the popularity of "YouTube" and "Videos" their benefits outweigh some of the disadvantages of screencasts over written documentation, such as content quality, convenience, version compatibility, and lack of searchability.

5.4.2 Future Directions for the Adoption of Screencasts

Our findings from the user survey provide insights on how users with different background and experience levels perceive various forms of documentation. The results from our survey also show the preferences in documentation resources among software engineers with different expertise. These insights can be used to recommend documentation resources that match more closely users' needs and preferences. For instance, depending on the level of experience, occupation, age, and problems being solved, the results from our survey can be used by project managers and organizations to include new media types (e.g., YouTube for less experienced professionals or less technical tasks and Stack Overflow for more experienced professionals or highly technical tasks) as part of the software lifecycle artifacts to improve the quality and usefulness of the overall system documentation and increase the productivity of their workforce.

In this survey, we specifically focused on how screencasts are used by our survey participants. Screencasts are the type of documentation that is perceived differently depending on the work and user context. As our findings show, there are several factors that can improve the usefulness of screencasts. Previous work [9] has identified best practices on how to create and design screencasts to document code. Our study is complementary to the previous work, since we are interested in identifying the purposes why users are watching tutorial screencasts and provided our participants with questions that consider a variety of non-programming and programming-related options to select from. Therefore, our findings are applicable to both types (non-programming- and programming-related) of software engineering screencasts.

Furthermore, our results support the observations and suggestions made by MacLeod et al. in [9] about designing short videos that are to the point [3], [9] and of high quality. In addition, our survey also highlights that screencast viewers would like to see screencasts to be complemented with other types of documents and that locating content and promoting videos are important issues for them. Based on our findings, we can extend the best practice suggestions made by MacLeod et al. in [9] to guide screencast creators and organizations to help improve the satisfaction of screencast viewers:

Build a dedicated platform: Currently there are paid[21] or unpaid[22] platforms [9] that provide content that is created by university professors or professionals from well-established companies. However, to serve a broader range of content creators and users (i.e., crowd), providing platforms dedicated to crowd-based software engineering tutorials, that replace YouTube which is used for multiple purposes, can provide better **monitoring and enforcement of quality standards** related to audio and visual quality of the videos. Such platforms can also support offline viewing of screencasts and provide a dedicated location for hosting specialized screencasts, which will improve accessibility and availability of screencasts. In addition, such platforms can also help

[21] Such as https://www.udemy.com/
[22] Such as http://mooc.org/

improving the overall content quality of hosted videos by promoting videos with high quality content and removing non-relevant videos.

Make content searchable and support annotations: As our survey showed, the content of a tutorial screencast is the main component why users select a particular screencast. As some of our survey participants stated, current video portals and players lack (semantic) search or index functionality, which would allow users to search for actual content or verify that the videos actually contain a certain content without having to play the whole video. In addition, current video players and portals lack the ability to annotate and save such annotated video content. Providing such advanced viewing features and annotation features would greatly benefit promoting further use of screencasts in the software engineering community.

Traceability links: Traceability links are not only an essential aspect in software engineering, but they can also be used to accommodating people with different documentation media preferences by identifying and providing relevant complementary documents to users. Providing traceability links between screencasts and other software artifacts can also address or highlight potential inconsistency of versions shown in screencasts and software artifacts and tools used by the viewers to avoid potential misleading or missing information. Integrating screencasts by linking them to other software and documentation artifacts can also benefit development processes where creating and updating formal documentation is not a priority. In this thesis, we address this issue of traceability links between GUI-based screencasts, linking an application feature with its relevant source code artifacts.

Consider target audience's level of expertise: As our survey results showed, people with different levels of professional experience use screencasts for a variety of purposes. One reason why software engineers do not watch tutorial screencasts is that they take more time and even sometimes are not useful. In addition, less experienced software engineers need more detailed explanation for a subject matter to fully comprehend it. The content of screencasts should therefore be adapted to the potential audience's level of expertise by: 1) adjusting the pace and describe or demonstrate a subject so that the audience can easily follow the information being presented in the videos (less experienced audience) or quickly allow them to find information they need (more experienced audience), 2) providing content with the necessary details in the same video or provide references to complimentary videos.

Promote videos for the right community: Our findings also illustrate that level of experience, occupation, and age of people does affect why users are viewing tutorial screencasts. Video creators should consider this information when promoting their videos or selecting portals where they publish their videos. For instance, visual learners (Gen Y and Z) and less experienced software engineers prefer to watch videos to learn new concepts. At the same time, experienced software engineers prefer Q&A sites more than videos to gain technical information. As a result, for screencasts covering non-technical topics, the target audience and therefore the content presentation should be geared more towards Gen Y and Z or less experienced viewers preferences, while for technical videos they may need to consider the level of experience of their target audience as well to make it engaging and understandable for them.

5.5 Threats to Validity

The goal of the presented survey is to provide insights on how and why screencasts are used by software engineers. The analysis of the responses from our user survey show that tutorial screencasts are popular information sources, especially among less experienced software engineers and that the content of a screencast is an important information source for many viewers when deciding which tutorial to watch. However, there are some threats to external, internal and construct validity regarding the survey and its results that need to be addressed in future work.

External Validity. In this study we received 99 responses by sharing our questionnaire on different social media platforms and mailing lists. In our study, most of the participants (around 60%) were graduate students. This is mainly because we shared the questionnaire on different social media and mailing lists as well as Moodle[23], which is only accessible by graduate and undergraduate students enrolled in a software engineering course. A potential limitation of our study is that our survey results may not be generalizable to non-students because of the selection bias, even though 55.5 % of the surveyed students had more than two years of professional experience. Future work needs to attract a more diverse set of participants and backgrounds, to improve the generalizability of the results and gain a better insight how screencasts are used among software engineers in an industrial setting.

[23] https://moodle.concordia.ca/moodle/

Construct Validity. We designed a questionnaire consisting of 34 questions and our main objective was to identify how and why software engineers use tutorial screencasts. For the survey, we used multiple choice, Likert scale, and open-ended questions to gather information from our participants. Being the work that tries to analyze how screencasts are used by software engineers, our questionnaire was able to capture and reveal useful results. Future work should provide facilities that allow for attracting software engineers to participate in a study that includes interviews, for additional clarification and follow-up questions, and/or allow for the inclusion of practical questions involving real use-case scenarios with different types of information sources. Such use-case scenarios would allow us to gain additional insights involving usability studies to observe how users are applying screencast tutorials or other types of documentation.

Internal Validity. Having mostly graduate students as our participants, resulted in having a large number of less experienced software engineers among our participants. Although our analysis results provide meaningful insights on how and why screencasts are used by software engineers, additional survey participants with more diverse backgrounds would be required to obtain a more representative sample and to increase the confidence in statistical significance of the study results.

5.6 Chapter Summary

In this chapter we conducted a survey to understand how and why software engineers use tutorial screencasts. We received 99 responses by sharing our questionnaire on different media platforms. Our findings revealed useful information and insights into the usage of screencasts for different purposes as well as what are other preferred media resources used for software development related tasks by our participants.

Among our main findings are that software engineers with less than two years of professional experience are the ones who rely more on “YouTube” or “Videos” to complete their software engineering tasks or learn new concepts/skills. Also, we found that the first choice of more experienced developers for getting help is “Question and answer sites” such as “Stack Overflow”. Our other finding is that screencasts are more used for getting help with non-technical tasks while “Question and answer sites” are more suitable to be used to fulfill technical tasks in software engineering.

We analyzed our received responses to better understand what the factors are why a person watches a tutorial screencast. Our findings show that the level of experience and occupation of a person can play a significant role in why they view tutorial screencasts to support their software engineering-related tasks.

In our study, we also investigated what are important information sources that our respondents consider when selecting a tutorial video to watch. Based on our findings, high quality content and having a narrator are among the most important criteria. Also, being able to search video content could vastly increase the accessibility and usefulness of screencasts as a source of software documentation.

Our open-ended questions revealed interesting and useful insights, highlighting strengths and weaknesses of screencasts compared to written documentation. We also received interesting suggestions on the role and usefulness of tutorial screencasts in software engineering domain. More specifically, based on our findings, while the content of screencasts is not easily searchable it is an important information source for users to choose a tutorial to watch. Furthermore, traceability between screencasts and complementary documentation (e.g., source code, other screencast tutorials, design documents, etc.) is identified to be useful for users.

As a result, we concluded with some recommendations for screencast creators and organizations who plan to integrate tutorial screencasts in the existing software engineering development practices.

Based on our findings and conclusions we found that there is a need for traceability between screencasts and technical software artifacts such as source code and provisioning the right format of documentation for all categories of users. This can facilitate software maintenance activities for developers and the ones who are passionate about open source contribution and fixing issues that are reported in GUI-based screencasts. As a result, we proposed a methodology for mining and linking screencasts that leverages audio and visual information (i.e., content) of screencasts. We describe our methodology in the next chapter.

6 Mining and Linking Crowd-based Screencasts

As our user survey in the previous chapter showed, 1) screencasts are frequently used, especially by less experienced developers, as an important documentation source, and 2) more experienced developers, if they watch screencasts, watch the ones that showcase application features (i.e., non-technical or GUI-based). Among one of the key challenges with current screencasts identified by our survey participants and in other work [1], [12] is that they lack traceability to other artifacts. The key benefits of providing such traceability (thoroughly discussed in Chapter 5) are: a) to allow to keep screencasts up-to-date with changes in the code and the version of a software application, b) to complement screencasts with other documents such as source code, which may make it easier for them to follow up on what they have seen on the screen or use a media type they prefer, c) to help development processes (e.g., agile) with automatically creating and updating complementary documentation.

Furthermore, while the content (i.e., visual or speech) and title of a tutorial video are identified as the most important information source for selecting which video to watch, currently video hosting platforms (e.g., YouTube) do not (fully) support content-based search, hence using the content of screencasts to establish traceability links is advantageous. We also found that providing access to the right format of documentation (i.e., textual, multimedia) for all categories of software engineers (e.g., highly experienced, less experienced) is another challenge that needs to be tackled.

As a result, our survey findings support our motivation (see Chapter 2) to introduce a methodology that supports extracting and mining information from screencasts that enables the provision of complementary type of documentation and the semantic linking (which considers the context in which keywords occur) of screencasts and their content to other software artifacts.

Figure 6-1. Methodology provides an overview of our mining methodology, which consists of identifying information sources and extracting relevant data from screencasts to be used for two main purposes: a) the mining of screencast information and content to enrich project documentation with complementary media types and therefore fulfill different users' preferences and, b) the semantic linking of this information (i.e., OCR text and speech content as source artifacts) to other artifacts (e.g., text extracted from relevant source code artifacts).

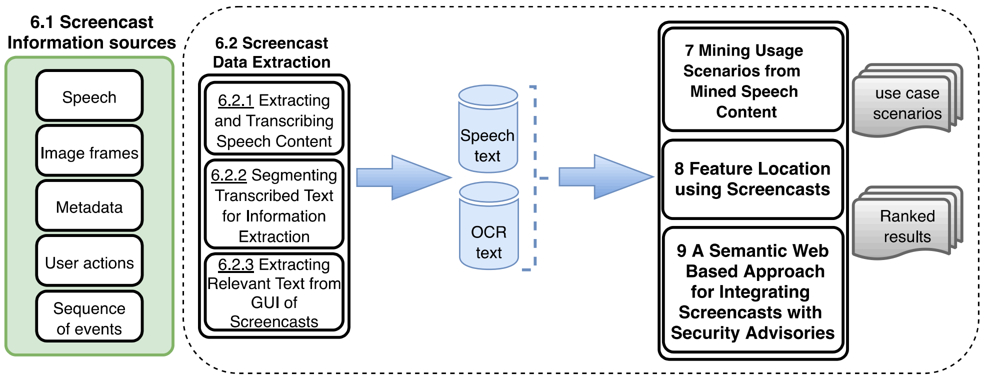

6.1 Screencast Information sources
Speech
Image frames
Metadata
User actions
Sequence of events
6.2 Screencast Data Extraction
6.2.1 Extracting and Transcribing Speech Content
6.2.2 Segmenting Transcribed Text for Information Extraction
6.2.3 Extracting Relevant Text from GUI of Screencasts
Speech text
OCR text
7 Mining Usage Scenarios from Mined Speech Content
8 Feature Location using Screencasts
9 A Semantic Web Based Approach for Integrating Screencasts with Security Advisories
use case scenarios
Ranked results

Figure 6-1. Methodology of mining and linking crowd-based tutorial screencasts.

6.1 Screencast Information Sources

In our approach, we extract data from screencasts to perform information extraction. Using data extracted from other types of software artifacts such as source code and security vulnerabilities, we can establish links between these artifacts and screencasts. Here, we focus only on screencasts that demonstrate how to use the GUI features of an application (see Chapter 2).

As explained by MacLeod et al. [2], there are various important elements of information inside screencasts:

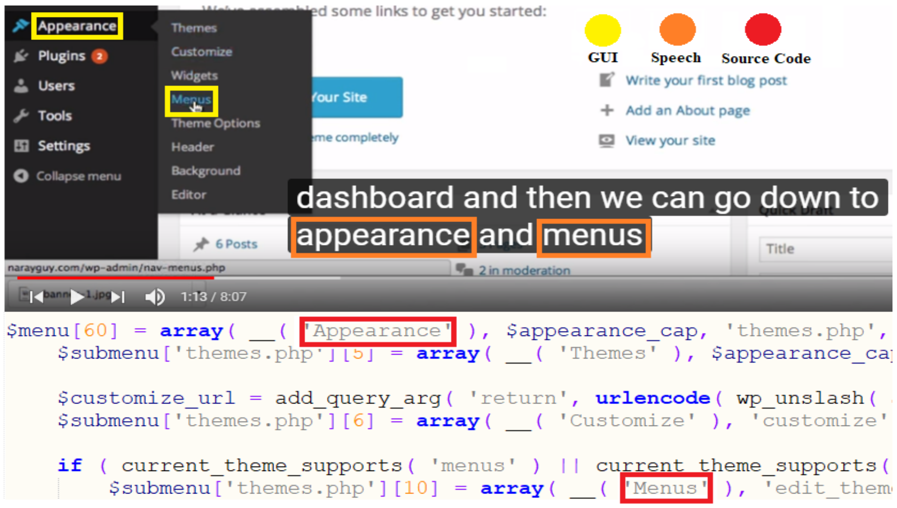
Speech: Tutorial screencasts often have a narrator who explains each step of a feature (use-case scenario), while demonstrating the feature on-screen. Parts of the speech not only will match labels or content in the image frames but likely also keywords found in other software artifacts. Figure 6-2 shows an example of such a keyword matching between screencast content (top) and source code (bottom).

GUI frames: Screencasts capture GUI interactions within a software application and command line terminal or IDE interaction, with each screencast containing a long sequence of images (frames) played at a constant frame rate. Since a typical GUI contains both textual and graphical information, the GUI text on a screencast frame could be matched to corresponding string literals in other software artifacts, unless the content is generated dynamically at run-time (e.g., user input). Furthermore, icons and graphical information such as edges, layout of the visual content, and color changes that occur in an image frame could be used as visual information clues to identify which feature aspect (or area of the screen) the narrator is currently focusing on.

User actions: A tutorial screencast includes textual, graphical, and speech information that dynamically changes as a result of a user's actions during the screencast (e.g., pressing a button, clicking on a menu item or opening another window, and inserting text). These actions are often closely related to corresponding software artifacts, such as GUI widget labels or event handlers in the source code or other artifacts such as bug reports, commit messages, etc.

Sequence of events: The order in which user actions occur during a screencast form a sequence of events that could correspond to a chain of event handler invocations, a method call graph, or a sequence diagram.

Metadata: When screencasts are uploaded to video portals like YouTube, they are typically annotated with some form of meta-and viewer-related data (e.g., title, description, upload date, comments, number of likes and dislikes, and other information related to the screencast). Such metadata provides additional information that can be used during the linking of screencasts and other artifacts such as source code content or vulnerability entities.



dashboard and then we can go down to appearance and menus

```
$menu[60] = array( __( 'Appearance' ), $appearance_cap, 'themes.php',
    $submenu['themes.php'][5] = array( __( 'Themes' ), $appearance_ca

    $customize_url = add_query_arg( 'return', urlencode( wp_unslash(
    $submenu['themes.php'][6] = array( __( 'Customize' ), 'customize'

    if ( current_theme_supports( 'menus' ) || current_theme_supports(
        $submenu['themes.php'][10] = array( __( 'Menus' ), 'edit_them
```

Figure 6-2. Matching keywords between speech, GUI and source code

6.2 Screencast Data Extraction

Screencasts are composed of an audio component and image frames. In what follows, we describe how we extract data from these two components and preprocess the data (Figure 6-3).

6.2.1 Extracting and Transcribing Speech Content

Tutorial screencasts usually have a narrator who explains the steps involved in performing a certain feature (scenario). The speech typically contains the rationale and insights of a screencast. For screencasts with closed captioning available, text extraction tools can be used to extract a screencast's subtitle. For screencasts published without closed captioning, automatic speech recognition tools to transcribe the speech information can be used. Since not all screencasts have closed captioning available, for all screencasts (published with or without closed captioning), we used automatic speech recognition tools to transcribe the speech information. Once the transcribed text is available, we remove stop words and for the remaining tokens, we perform stemming to prepare the data for extracting use-case scenarios or using it for source code/vulnerability localization.

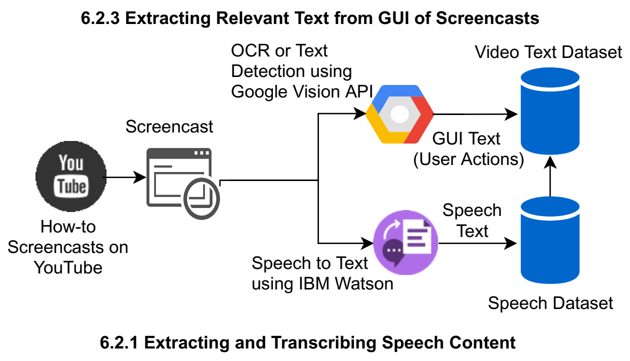

6.2.3 Extracting Relevant Text from GUI of Screencasts
OCR or Text Detection using Google Vision API
Video Text Dataset
Screencast
GUI Text (User Actions)
You Tube
How-to Screencasts on YouTube
Speech Text
Speech to Text using IBM Watson
Speech Dataset
6.2.1 Extracting and Transcribing Speech Content

Figure 6-3. Screencast data extraction.

Before any analysis can be performed, we first need to extract the spoken text of the video as a speech file using standard speech extraction tools.

Given the speech part of a video, we then need to transcribe the downloaded speech files using a Speech-To-Text (STT) tool to obtain an automatic transcription of the spoken text (Figure 6-3). The availability of an STT transcription tool that can produce outputs that are accurate enough and have a low enough error rate is an essential requisite. While transcribing speech files in general is a challenging aspect, most STT tools can support some standard English dialects (mostly native US or UK English). To improve the transcriptions of spoken text, the majority of existing STT tools (e.g., Dragon NaturallySpeaking[24]) rely on an initial training step, during which the tool can be trained to recognize a particular voice (dialect) of a user.

However, transcribing speech content from crowd-based video portals introduces additional challenges. Among these challenges are the fact that videos (and therefore also their speech components) are created and published by many different users with cultural, intellectual, communication and linguistic differences. While processing crowd-based speech sources (where by definition the presenter is not one single person), STT tools will have to deal with the resulting variety of English dialects (e.g., native speakers, non-native speakers with accents or grammatical errors). Unfortunately, a training phase (as Dragon is doing) is not feasible since the videos are crowd-based. To deal with this uncertainty, some tools provide as part of their transcription also indicators about the accuracy and quality of their transcription. Given that the quality of the STT output affects our methodology, we eliminate transcriptions that are flagged by an STT tool to be of potentially low quality using measures such as a confidence score that is a value of accuracy that STT tools provide.

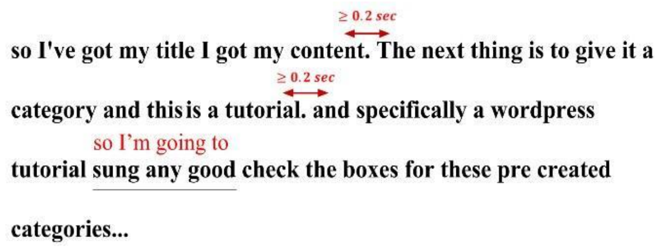
6.2.2 Segmenting Transcribed Text for Information Extraction

Another major challenge in STTs besides grammar and dialect, is the need to extract punctuated sentences instead of a bag of words to be able to perform information extraction on speech data. Punctuation, often also referred to as sentence segmentation, is an essential aspect to allow for further processing of the transcribed text through Information Extraction (IE) systems or libraries (e.g., GATE[25], OpenNLP[26], NLTK[27]). IE systems analyze text to extract information about pre-specified types of events, entities or relationships, where the order in which the information is presented matters. To be able to extract this information, IE tools require punctuation and newline characters for sentence splitting and text segmentation.

Nevertheless, STT tools provide only very limited support for punctuation in their transcriptions, mostly relying on fixed time thresholds for speech pauses. However, without taking into consideration a user's specific speech and speed patterns, sentence segmentation of spoken text

[24] https://shop.nuance.com/
[25] https://gate.ac.uk/
[26] https://opennlp.apache.org/
[27] https://www.nltk.org

becomes inherently difficult due to the incorrect punctuation in the transcribed text resulting often in no logical sentences. Furthermore, users tend to use informal grammar and sentence structures when narrating their videos with spoken text. This and any grammar and spelling issues generated by incorrect transcription further reduces the chances to correctly identify sentence boundaries. Figure 6-4 illustrates these challenges.


≥ 0.2 sec
≥ 0.2 sec
so I'm going to

so I've got my title I got my content. The next thing is to give it a category and this is a tutorial. and specifically a wordpress tutorial sung any good check the boxes for these pre created categories...

Figure 6-4. (a) Threshold based sentence segmentation based on speech pauses. (b) Imprecisions caused by grammar and transcription errors.

In our methodology, we take advantage of timestamps commonly provided by modern STTs for each word in the transcribed text, then calculate the pauses, considering the baseline approach in [82] and their *mean* and standard deviation (*sd)* values for each file to adjust the punctuations based on the speaking pace using these non-acoustic features. As part of an initial experiment of using different numbers of α, we observed that using an adjustment factor $\alpha=3$ for the *sd* produces the best sentence segmentation results:

$$mean + \alpha \times sd \qquad (6\text{-}1)$$

6.2.3 Extracting Relevant Text from GUI of Screencasts

To extract text from video frames, one needs to first extract image frames of the screencasts, then extract text fragments from them that are relevant to the use-case scenario that is being presented in the screencast. Such relevant text fragments typically are either found in the image frames or in text/labels associated with them. Given the many frames within a typical video, only those in which a major event (e.g., mouse click or pressing a button) happens, i.e., key image frames, should be targeted.

Unfortunately, identifying these text fragments and frames is quite challenging. We experimented with three different image processing approaches, before settling with a simpler, textual approach. First, we used Template Matching [83], where one should provide a template image of a mouse pointer (both the clicking and idle version) to be able to automatically locate the mouse pointer in video frames and any mouse actions being performed. We also experimented with image pixel subtraction [84], which subtracts the pixels of neighboring frames to detect changes occurring between image frames, typically caused by mouse movements. Finally, we also used connected components detection [85], which exploits the fact that neighboring pixels typically have similar pixel values to locate different logical areas in the image frames (e.g., a "text field", or "button").

However, these image-processing approaches have several drawbacks, namely processing overhead, strong dependence on image quality or image resolution, and possible information loss due to image binarization or transformations. We also experimented with both the transformation of images to gray scale and their binarization to: 1) reduce the dependency of our approach on background color, 2) to improve the recognition of text and icons, and to 3) reduce the computational resource overhead by reducing the image dimensions. However, these transformations caused areas with light background and white text to vanish. In addition, our template matching technique did not perform well due to users customizing their mouse pointers and mouse pointers often being hidden by labels during the mouse click.

Instead, we opted for a much simpler, pure textual approach to detect features during a screencast. For the feature detection we use Optical Character Recognition (OCR) [86], which we applied on image frames to recognize text shown in the frame (Figure 6-3). We select every two subsequent image frames based on their order in the image frames that are extracted from a video and we then subtract the text (instead of the pixels) of these 2 subsequent image frames:

$$
\begin{aligned}
&Diff(i) = Text(img_i) - Text(img_{i-1}) \\
&Diff = [Diff(2), Diff(3), \dots, Diff(k)] \\
&Diff_{final} = [d_i |\ d_i \in Diff \ \wedge \ |d_i| > 0]
\end{aligned}
\tag{6-2}
$$

where k is the number of the image frames of a screencast. OCR returns all text enclosed within neighboring pixels on the images along with its coordinates in the images. An advantage of this approach over image processing approaches is that during the analysis only a bag of words is used and therefore significantly reduces the processing overhead. For example, if a user clicks on a button, the GUI will change to open a dialog menu or new text field, which will be reflected as a change in the text recovered through OCR. $Diff(i)$ contains the bag of words (Figure 6-5) that are added/modified in each subsequent image frame (i.e., $Text(img_i)$) and are considered to be relevant words to the scenario. Our screencast data sets in which each screencast becomes a text document contains only the $Diff_{final}$, which is all the text derived from extracting GUI text difference (derived using our formula 6-2) between successive image frames that are extracted at a specified frame rate.

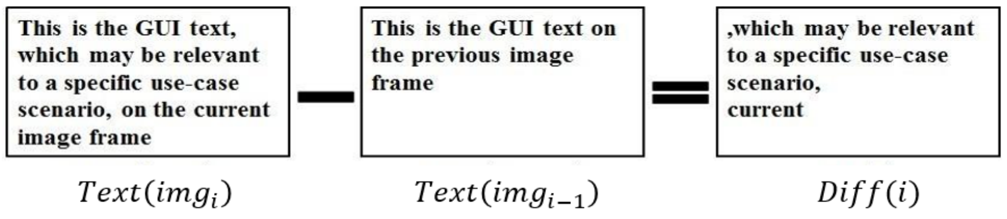

This is the GUI text, which may be relevant to a specific use-case scenario, on the current image frame
This is the GUI text on the previous image frame
,which may be relevant to a specific use-case scenario, current
$Text(img_i)$
$Text(img_{i-1})$
$Diff(i)$

Figure 6-5. GUI text difference after a user action happens.

6.3 Linking Screencasts to other Artifacts

So far, the output of this approach is mined speech/GUI content that can be used for information extraction and linking purposes. In the final step of our approach, depending on the target application we apply algorithms and technologies that enable semantically linking screencasts to other artifacts to our preprocessed data. In different applications of our methodology, 1) we leverage LDA (see Section 3.2) to perform information extraction on speech data (Chapter 7), and 2) to perform source code feature location (Chapter 8), and 3) we use a Semantic Web-based approach to establish links between vulnerability exploitation screencasts and their relevant vulnerability descriptions on NVD (Chapter 9). The input to these algorithms is our source artifact which is screencast data (speech and OCRed text) and our output is software artifacts such as use-case scenarios, source code, and NVD descriptions.

6.4 Chapter Summary

In this chapter, we presented an approach for mining the content of crowd-based software engineering tutorial screencasts and linking them to other software artifacts such as source code. Our approach allows us to enrich current software documentation by mining speech component of screencasts. It also provides the basis for the linking and integration of crowd-based screencasts with other software artifacts by using speech and textual cues of screencasts. In the following chapters we describe three applications of our proposed methodology. We present how we applied our proposed methodology to extract use-case scenarios from the speech of screencasts. We also show how we link screencasts to source code implementation of a feature that is being shown in them. Finally, we present a Semantic Web-based approach that leverages our methodology to link screencasts that demonstrate a vulnerability explosion to their relevant entities on NVD.

7 Mining Crowd-based Speech Documentation

7.1 Introduction

This chapter instantiates our methodology introduced in Chapter 6, to automatically transcribe and analyze the content of the transcribed text using IE techniques to extract usage (i.e., use-case) scenarios from the speech part of screencasts since those typically contain the rationale and major insights about what a video is trying to achieve, complementing the visual cues. Our goal is to both supplement existing software documentation and to help users focus on the relevant sources. We aim to extract textual information (i.e., use-case scenarios) from speech component (i.e., audio) of screencasts to accommodate users who prefer written documentation. Furthermore, a key benefit of written documentation is being searchable since content searchability is declared by our survey participants (see Chapter 5) to be one of the advantages of written documentation over screencast documentation.

As part of our evaluation, we present a case study in which we extract use-case (or usage) scenarios from the speech of WordPress tutorial videos. For the second part of our case study, we evaluate how well existing metrics of videos are able to rank videos according to relevancy for a given feature or use-case. This study extends traditional MSR approaches to include the mining and analysis of speech components of video material related to software products. A further contribution of this work is to provide an approach that allows the automated extraction of software documentation from speech content. The presented case study illustrates not only the feasibility but also benefits of mining the speech component of videos and how users in a software development or maintenance context can benefit from this “wisdom” of the crowd.

In our case study, we will be answering the following two research questions:

- **RQ5**. How accurately our automated approach to extract software engineering related information from the speech part of crowd-based videos is able to enrich existing documentation?
- **RQ6**. How well do existing screencast ranking mechanisms perform in terms of relevancy of the mined speech content?

In fact, in RQ5 we will evaluate how well our approach is able to extract and recover use-case scenarios from the speech part of how-to screencasts published on YouTube. For RQ6, we rank screencasts based on how complete their coverage of a given use-case scenario is (as mined in RQ6), then evaluate which video- related metrics provide the closest ranking.

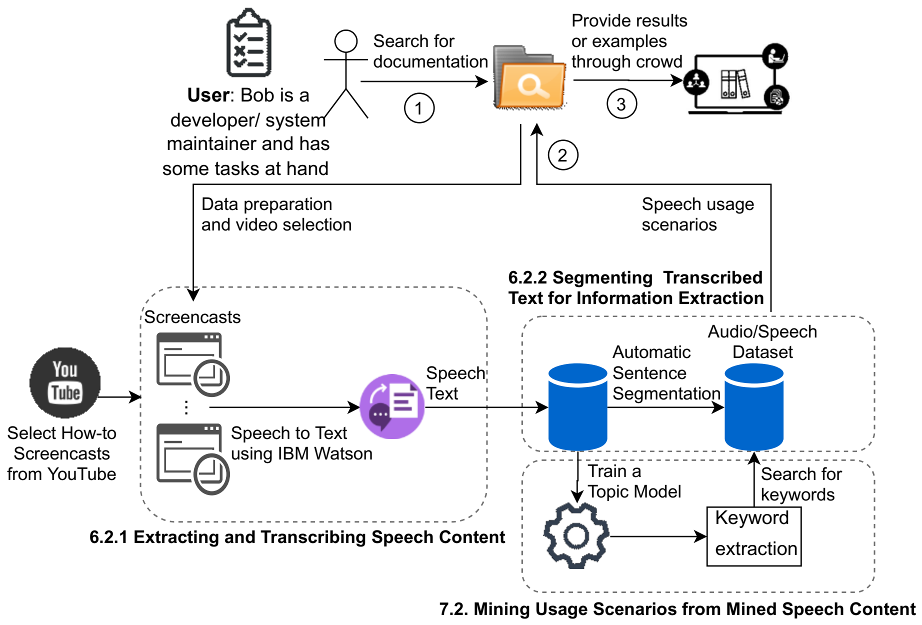

User: Bob is a developer/ system maintainer and has some tasks at hand
Search for documentation
1
Provide results or examples through crowd
3
2
Data preparation and video selection
Speech usage scenarios
6.2.2 Segmenting Transcribed Text for Information Extraction
Screencasts
Speech Text
Audio/Speech Dataset
Automatic Sentence Segmentation
Select How-to Screencasts from YouTube
Speech to Text using IBM Watson
Train a Topic Model
Search for keywords
Keyword extraction
6.2.1 Extracting and Transcribing Speech Content
7.2. Mining Usage Scenarios from Mined Speech Content

Figure 7-1. The proposed methodology to mine the speech part of screencasts.

Figure 7-1 illustrates an application of our proposed methodology for mining crowd-based speech documentation. In what follows we provide the details of applying our methodology to extract use-case scenarios from speech of screencasts.

7.2 Mining Use-case Scenarios from Mined Speech Content

To extract use-case steps from speech, one should first select a proper IE technique. Analyzing speech data shows that there are two types of use-case steps: 1) Essential use-case steps by definition should be common to all transcribed videos. 2) Optional use-case steps should be specific to a subset of transcribed videos (Figure 7-2). This means that in principle we could apply any IE techniques such as, text classification or clustering on the corpus of transcribed speech [87], to extract actual use-cases from the screencasts.

1. Click on the “new” button
2. Type the title of the blog post
3. Change the font
4. Type the content of the blog post
5. …

Figure 7-2. Sample use-case steps of "how to create a post in WordPress". The optional step is marked with underline.

However, common IE approaches for textual data such as dependency parsing [88], [89] or language models [87] are not suitable for our dataset, since the grammar and syntax of sentences in spoken text differ significantly from those of traditional written text and there is not enough domain-specific training data for developing a language model. This situation is further complicated by additional errors introduced by STT tools, during the transcription of the spoken text.

Based on our review of existing IE approaches, we selected topic modeling [90] to identify the topics that are shared amongst all videos for a given system feature. Given that all essential steps of a given use-case should be mentioned by each screencast related to that use-case, we conjecture that there should be at least one dedicated topic (group of words) containing all essential steps.

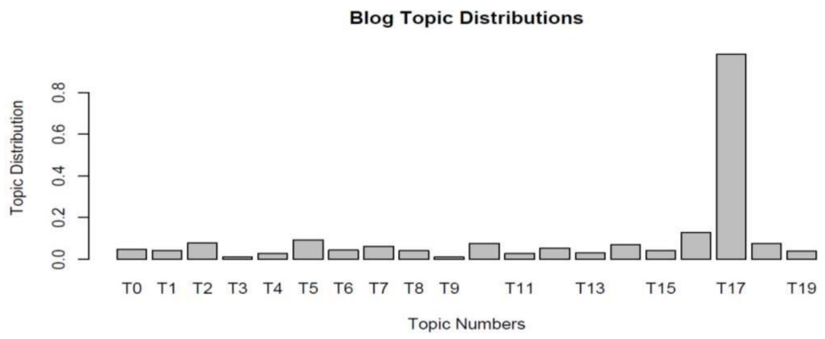


Figure 7-3. Distributions of topics for each topic model for the create blog topic

For example, Figure 7-3 shows the distribution of the group of words describing a topic across the transcripts of the use-case “Creating a Blog”. A higher value for “Distribution” means that the topic occurs in more transcripts and hence is shared by more videos. We would automatically select for each use-case the topic with the highest distribution value (in this example it would be topic *T17*). That topic is assumed to contain all essential steps and some of the optional steps.

now when I point to the post **button** it gives me little fly out menu.
and I'm gonna **click** on **add** new it's something you'll get familiar with because this will be one of the most common activities for you inside of **WordPress** so here's the post.

Figure 7-4. Sentences with topic keywords (highlighted), corresponding to essential steps of a use-case.

Finally, in order to reconstruct the individual use-case steps from the selected topic, we first identify the 20 words of the topic that are the most common in the analyzed screencasts. These represent the topic's "keywords". We then use regular expressions to identify in each screencast's transcription all sentences containing these keywords (Figure 7-4). For a given screencast, the corresponding sentences are the essential and optional use-case steps discussed by the screencast. Other sentences can be filtered out.

Since each screencast results in a sequence of extracted use-case steps all that is left is to rank the resulting screencasts based on the relevance of these steps. Coming up with a novel ranking mechanism is outside the scope of this work, instead in our case study (Section 7.4) we will evaluate existing ranking mechanisms in terms of relevancy.

7.3 Proof of Concept

To illustrate the applicability of our approach, we introduce a proof of concept implementation of our methodology, which we then use in the next section for case studies to address our two research questions introduced earlier in this chapter. The following sections describe in more details technical and implementation considerations for creating our proof of concept implementation.

7.3.1 Data Preparation and Transcribing Screencasts

The data sources used by our approach can be any podcast or screencast that targets a specific software project or product feature. Such videos or podcasts typically can be found on YouTube, Vimeo or on a project's wiki or web site. The videos usually come in the form of how-to or walkthrough tutorials, workarounds for known bugs or describes implementation issues related to a particular project. It should be pointed out that any video source would be applicable as long as it has a speech component with limited background noise (e.g., music, environmental noise)

For the automated transcription of speech content of videos, we had to select an STT tool that was suitable to deal with the challenges associated with analyzing speech content from video material created and published by many different users. For the tool selection process, we used several evaluation criteria, including: precision, recall, support for different English dialects (e.g., US, UK and Indian English dialects), scalability, provision of metadata (e.g., timestamps, accuracy score for transcript), and ease of use. As evaluation data we used one screencast, which we manually transcribed to establish a baseline.

Table 7-1. Comparison of the output of STT tools using British dialect, length of the transcript (in terms of characters), precision, recall and f-measure.

Tool	Length	Precision	Recall	F-Measure
Dictation[28]	5376	0.75	0.66	0.70
Google API[29]	6458	0.75	0.75	0.75
IBM Watson[30]	7745	0.75	0.88	0.81
Speechlogger[31]	6971	0.74	0.73	0.73
Speechpad[32]	6415	0.70	0.74	0.72

Table 7-1 shows the five analyzed STT tools, consisting of commercial products by top vendors in the field, as well as how they performed on our evaluation. First, we manually transcribed the screencast, then determined for each tool how many words they yielded in their transcription, how many of those words were correct (precision), and how many of the manually identified words they were able to identify (recall). The F-measure then provides a combined measure of precision and recall, the higher the value the better both precision and recall are. We found that IBM Watson obtained the best recall (more words transcribed correctly), while precision was similar to the other tools. Hence, we decided to use IBM Watson.

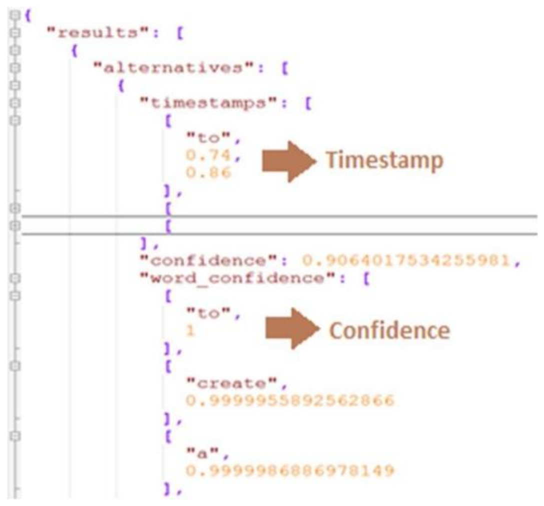
Note that our evaluation did not consider the order of words into sentences. However, as shown in Figure 7-5, the sample JSON output of the IBM Watson STT provides metadata that is useful

[28] https://dictation.io/
[29] https://www.google.com/intl/en/chrome/demos/speech.html
[30] https://speech-to-text-demo.mybluemix.net/
[31] https://speechlogger.appspot.com/en/
[32] https://www.speechpad.com/

for the later processing steps, like sentence splitting and filtering of the transcripts. This was another major reason to opt for this STT tool.

In the JSON file, *Word confidence* corresponds to the STT's confidence level, according to which the tool estimates whether the transcribed words were correctly recognized. Based on manual analysis of the transcripts generated by the STT tool, we observed that a transcript with an overall score below a threshold of 0.7 contains too many errors to be useful for further processing. For our proof of concept implementation, we therefore eliminated all transcripts with a confidence score below the 0.7 confidence threshold.

While the IBM STT tool provides some very basic sentence splitting, this resulted in very few, extremely long sentences. Therefore, we performed the additional sentence splitting processing of formula (6-1). For this splitting process, we took advantage of the "timestamps" meta-data provided in the output of the IBM Watson STT. The tool provides the start and end time of each word in the speech file's transcription, which allows us to calculate the pauses between the words in the spoken text. We use these pauses in connection with the sentence splitting formula (6-1) to calculate the *mean* and *std* values of the pauses, for each file. Using this approach for sentence splitting, the number of sentences increases on average by a factor of 5.



```
{
  "results": [
    {
      "alternatives": [
        {
          "timestamps": [
            [
              "to",
              0.74,
              0.86
            ],
            [
            [
          ],
          "confidence": 0.9064017534255981,
          "word_confidence": [
            [
              "to",
              1
            ],
            [
              "create",
              0.9999955892562866
            ],
            [
              "a",
              0.9999986886978149
            ],
```

Figure 7-5. Sample JSON output created by the IBM Watson STT tool.

7.3.2 Information Extraction

To identify topics across the transcripts, we used the standard MALLET[33] tool for topic model analysis. MALLET is a popular open source tool that is used by other research in software engineering doain [32]. We evaluated different configuration settings, with the number of topics set to 5, 10 and, 20 topics for each use-case and manually compared the granularity of the results. The analysis showed that, for our purposes, the configuration with 20 topics produced the best results. As a result of the extraction process, we then chose the topic with the highest distribution value across the corpus of screencasts of the use-case under study.

7.4 Case Study

In what follows, we present results from a case study that we conducted to evaluate the applicability of our methodology and to answer our two research questions introduced earlier.

First (RQ5), we evaluate how well a fully automated approach using IBM Watson STT performs. Then (RQ6), we evaluate whether existing screencast-related metrics would be good indicators of the relevancy of a screencast towards a specific use-case.

Dataset: For our case studies, we created a dataset based on how-to screencast videos for the WordPress CMS, published on YouTube. We selected WordPress as our study subject, since it is one of the most widely used, mature and documented open source CMS tools. Furthermore, given the popularity of WordPress and its open source nature, a large number of how-to screencast videos exist on YouTube, covering most of the WordPress use-cases and features. As our ground truth we use 5 topics selected from the WordPress online documentations and made them publicly available.

Screencast selection was based on five keyword search queries (obtained from the official WordPress documentation) that we performed on YouTube: 1) "*How to create a blog post in WordPress*" (blog), 2) "*How to change the font in WordPress*" (font), 3) "*How to password protect WordPress*" (password), 4) "*How to create a Gravatar in WordPress*" (gravatar) and, 5) "*How to add a feed to WordPress*" (feed). We selected videos from top results and the recommendations

[33] http://mallet.cs.umass.edu/

made by YouTube. The length of the obtained screencasts varied from 1 to 10 minutes. Table 7-2 provides a summary of the datasets that we created for each of the queries. Of the initially downloaded screencasts, between 60% (blog and feed) and 83% (password) of the screencasts yielded a useful transcript with STT confidence of 0.7 or higher.

Table 7-2. The number of studied screencasts and transcripts for the five analyzed use-cases

Category	**# screencasts**	**# transcripts**	**# transcripts filtered by confidence**	**% transcripts filtered by confidence**
Blog	42	35	25	59.5
Font	27	21	21	77.8
Password	23	23	19	82.6
Gravatar	30	29	21	70
Feed	30	29	18	0.6

RQ5. How accurately our automated approach to extract software engineering related information from the speech part of crowd-based videos is able to enrich existing documentation?

Approach: After transcribing the downloaded screencasts (see Table 7-2), we experimented with different configurations of MALLET using 5, 10 and 20 topics. Table 7-3 shows, for each configuration, the maximum value for topic distribution. The higher this value, the more fine-grained the topics tend to be (covering less files). Here, we are interested in higher values, however too high values might indicate too fine-grained topics. After manually checking the maximum distribution topics of the different configurations, we found that using 20 topics, we can have the maximum distribution values, as well as topics that are not too fine-grained or too coarse-grained for our purposes. Hence, we used 20 topics in our case study.

Using the 20 topics setting, we selected, for each of the five use-cases, the topic with the highest distribution value across all analyzed screencasts. If there are other highly distributed topics, we would pick the highest one. Figure 7-6 shows these distribution values of the 20 topics for each use-case. For example, for the *blog* use-case, we would pick topic 17 as the one containing the most common topic keywords across all video transcripts for this use-case. Using regular expressions, we then parsed the generated transcripts and extracted those sentences that contain at least one relevant topic keyword. Each of these extracted sentences corresponds to one use-case step for this feature.

Table 7-3. Topic distributions for each feature/topic model.

Category	Maximum topic distribution		
	5 topics	**10 topics**	**20 topics**
How to create a blog post in WordPress	0.86	0.96	**0.98**
How to change font in WordPress	0.37	0.76	**0.98**
How to password protect WordPress	0.38	0.53	**0.63**
How to create a Gravatar in WordPress	0.39	0.26	**0.94**
How to add a feed to WordPress	0.33	0.66	**0.72**

The processing time for the 5 datasets varied between 487.5 seconds for the *Password* dataset and 809.5 seconds for the *Blog* dataset. A detailed breakdown is shown in Table 7-4, which is the processing times based on a single run of each step of the methodology.

Table 7-4. Detailed breakdown of processing times (seconds).

Dataset	**STT**	**Topic Modeling**	**Regular Expression + JSON parser**
Blog	790	1.5	18
Font	773	1.5	18
Password	458	1.5	28
Gravatar	582	1.5	30
Feed	683	1.5	19

Finally, based on the existing WordPress documentation, we manually derived a ground truth for each use-case, i.e., the sequence of essential and optional steps that make up the use-case. We validated these use-cases by running them on an example installation of WordPress. Given these ground truth steps, we then manually calculated the precision, recall and F-measure of our approach. Precision in this context corresponds to the percentage of selected sentences in the transcripts that also occur in the ground truth, while recall refers to the percentage of steps in our ground truth that were found by the approach. The F-measure then is the harmonic mean of precision and recall, the higher it is, the higher the combination of precision and recall are.

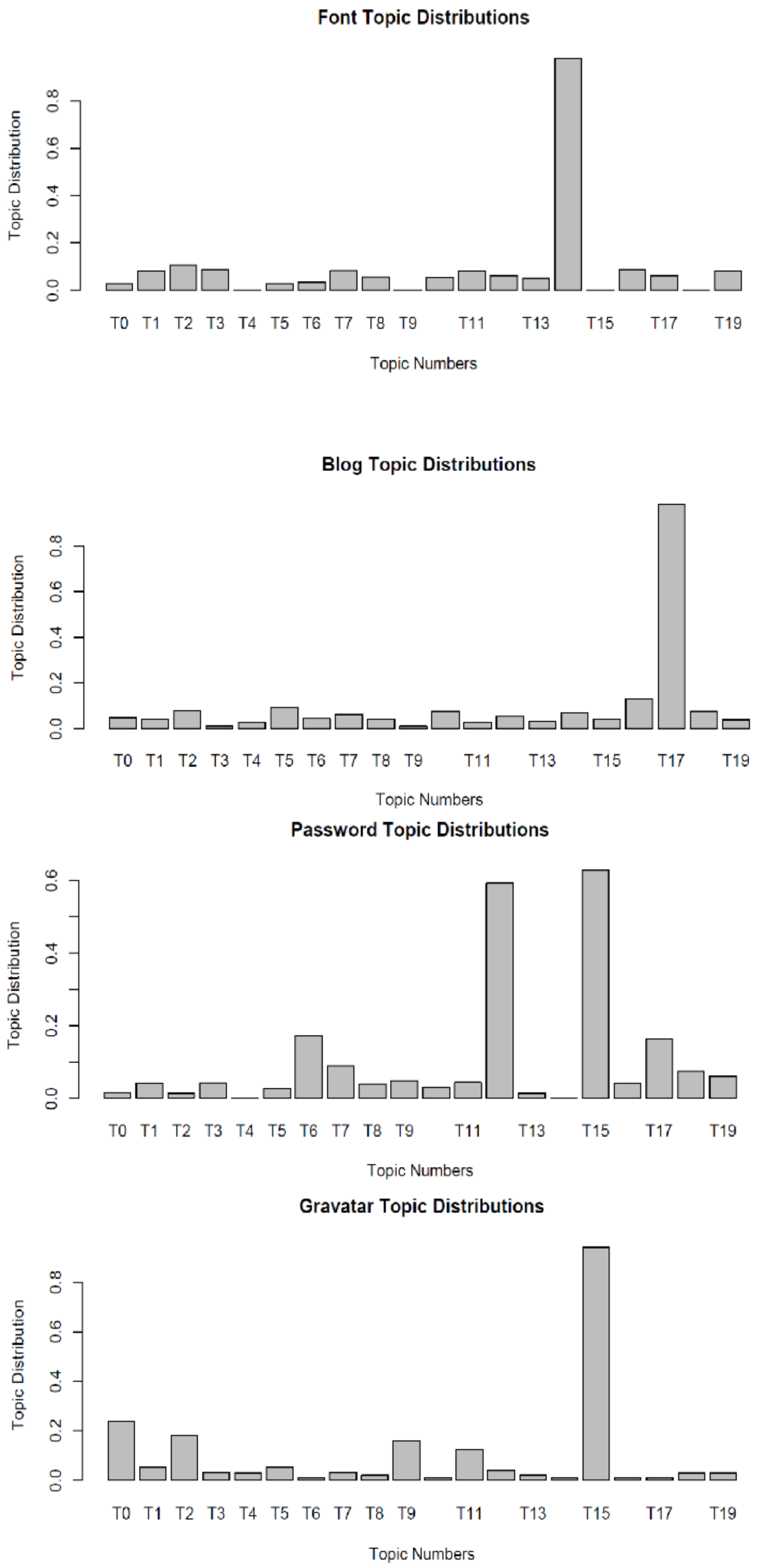


Figure 7-6. Bar charts showing the maximum distribution of topics for each topic model and use-case data set.

Since evaluation of precision and recall required manual comparison of the ground truth and the mined use-case steps (which is time consuming), we randomly selected 5 screencasts for each use-case, making sure to include screencasts with many, few and a medium number of views. While we only have precision, and recall for the resulting 25 screencasts, the topic models have been built based on all screencasts. Furthermore, note that we did not make a distinction between essential and optional steps for our calculations; hence missing optional steps will reduce the recall of our approach. This strict evaluation means that the obtained recall values are only a lower bound (ignoring optional steps would increase recall, and slightly reduce precision). Furthermore, we did not evaluate whether the mined use-case steps occurred in the right order. From our informal evaluation, we observed that the order typically was correct.

Table 7-5 shows the corresponding precision, recall and F-measure values for the 25 manually evaluated screencasts.

Findings: Precision and recall both are high (median of 83.33% and 100%, respectively). Table 7-5 Precision (%P)/Recall (%R)/F-measure (%F) of the 25 manually evaluated transcripts.Table 7-5 shows how the minimum precision value is 62.5% (Font), with most of the values 100%, implying that most of the use-case steps suggested by the approach are correct (low number of false alarms). Recall, on the other hand, sometimes drops to 50%, meaning that only half of the relevant information (i.e., optional and essential steps) is recovered from a screencast. This may be the result of narrators using different words for the same action in a specific use case step. Overall, however, the 25 screencasts have a median “F-measure” value of 84.32%, indicating a very high performance.

Closer analysis of the use-case steps produced by the approach showed that almost each screencast started with an introduction giving an overview of the content of the screencast and ended with a recap of the video. Both intro and recap were selected by the approach as being part of the use-case steps, while our ground truth did not include it. Filtering out those sentences (typically at start/end of recovered use-case steps) would increase the median precision to 100%.

For each use-case, at least one of the screencasts achieves a recall of 100%. In such cases, the approach retrieves all the relevant steps (i.e., true positives) and has no false negatives. This means that, theoretically, if one could somehow rank screencasts according to their predicted recall, users could just select the obtained use-case steps of the top ranked screencast. If we assume such a ranking to exist, we could conclude that crowd-based speech documents would be a reliable source to enrich software documentation. RQ6 explores this in more detail.

Table 7-5 Precision (%P)/Recall (%R)/F-measure (%F) of the 25 manually evaluated transcripts.

Blog			Font			Password			Gravatar			Feed		
%P	**%R**	**%F**	**%P**	**%R**	**%F**	**%P**	**%R**	**%F**	**%P**	**%R**	**%F**	**%P**	**%R**	**%F**
75	60	66.67	100	67	80.24	100	100	100	100	75	85.71	100	67	80.24
100	100	84.19	80	100	88.89	67	67	67	100	100	100	100	100	100
100	100	100	100	100	100	75	100	85.71	71.42	83	76.78	85.7	100	92.3
100	50	66.67	83.33	100	90.91	67	100	80.24	82.7	86	84.32	75	100	85.71
100	50	66.67	62.5	100	76.92	100	75	85.71	66.87	67	66.93	71.4	83	76.76

Findings: Across the 25 screencasts, our approach reports a median of 23.08% of the transcript sentences as being use-case steps. Given the high precision and recall of the approach in identifying use-case steps, this low number of transcription sentences being required for describing these use-case steps is a good indicator that our approach does also perform well in summarizing the content of screencasts.

Conclusion: The approach can correctly extract most of the use-case steps. Also, since for each use-case there is at least one screencast whose relevant steps are retrieved, use-case scenarios that are extracted from crowd-based screencasts can be used as a supplementary source of documentation by software developers. Further analysis shows that the proposed approach is potentially able to summarize the spoken text of screencasts by filtering out the sentences that are not among use-case steps.

RQ6. How well do existing screencast ranking mechanisms perform in terms of relevancy of the mined speech content?

Approach: Existing search engines of online video portals, such as YouTube, use different criteria for ranking their result sets based on a user's search query. Most of the criteria are based on the similarity of the search keywords, relying on the textual description provided by the user (e.g., title, tags or video description), as well as the number of views or user ratings. However, none of these search engines analyzes and takes advantage of the actual speech content (image frames or speech) in their ranking. The closest match is Google's STT, whose results (bag of words) are added to the textual description of a video. However, the actual meaning of the video in case of a screencast, i.e., the key steps to follow, are not considered.

It is not uncommon that videos with low quality content (e.g., not related to top of interest, or with an incorrect title) might be ranked first. In addition, criteria such as number of views might not always be a reliable indicator, since in some cases videos might go “viral”, due to some publicity or a particular event, even if their content is not necessarily of high quality or relevant.

Future work is required to derive an optimal ranking algorithm for use-case mining of screencasts. However, as a first step in this direction, we use the evaluation results of RQ5 and determine, per use-case, optimal relevancy criteria for the 5 screencasts. We then compare the relevancy of the transcriptions to the following criteria: length of the screencast (measured by the number of words in the transcript), number of topic keywords appearing in a transcript, confidence score reported by the STT, reported feedback on the video (total number of likes and dislikes), and number of views of the video online. We define the optimal ranking as the one based on recall, with the idea that the most complete screencast in terms of use-case steps is the most desirable one.

Given that, per use-case, we only have recall values of 5 screencasts, our findings are preliminary. However, since we have 5 different use-cases, we should be able to notice any trend. We use Spearman rank correlation, given the small sample size. Note that we cannot combine the 25 screencasts in one data set and calculate one correlation value, since some screencasts are in general more popular than others or require longer explanations, which would lead to incorrect correlation values. Table 7-6 contains the resulting correlation values.

Findings: The number of topic keywords in a transcript, and the STT confidence currently seem to be the best indicators of relevancy. Unsurprisingly, video length also has a high correlation, since the more one says in a screencast, the higher the probability to cover an additional use-case step. Hence, this correlation is not that useful in practice. The fact that high confidence correlates with higher recall also might be by design, as our approach only keeps high confidence screencasts. On the other hand, if a topic model is built on a set of screencasts for a particular use-case, any additional screencast for that use-case might leverage the existing topic model such that a ranking based on number of topic keywords becomes feasible.

On the other hand, the number of views is not a reliable ranking, since correlations vary from very high, positive correlation to very high, negative correlations. This could confirm our remarks about viral videos or other sources of noise in this metric. Finally, feedback is also unreliable, giving equally extreme correlation values. Future work should try and further improve the current rankings.

Table 7-6. Correlations between transcript properties, for each use-case.

Dataset	Recall-Length	Recall-Keywords	Recall-Confidence	Recall-Feedback	Recall-Views
Blog	0.81	0.89	0.79	0.65	0.63
Font	0.71	0.71	0.71	0	0
Password	0.67	0.89	0.89	-0.11	-0.78
Gravatar	0.60	0.70	0.70	-0.87	-0.40
Feed	0.34	0.22	0.45	-0.67	-0.67

***Conclusion*:** The evaluations show that the optimal ranking criteria for the 5 use-cases studied here differ from those that are usually used by video search engines. The content of the videos, especially the number of keywords that are relevant to the use-case topic is the best relevancy indicator for the studied screencasts and use-cases.

7.5 Threats to Validity

This chapter presents an application of our methodology that is taking a step towards making relevant content from screencast tutorials, created by the crowd, become an integrated part of existing software documentation. Our case studies illustrate that our methodology can deliver on these objectives. However, the data set and methodology are very different from existing approaches, leading to some challenges that might affect our reported results.

External Validity. In this work, a large, but still limited number of documents related to five use-case scenarios of WordPress (a mature and widely used CMS) are analyzed. Therefore, to be able to generalize this approach and prove its applicability in different domains, it should be applied on larger datasets extracted from different domains and for extracting different types of documentation. While analyzing a larger number of speech files would increase the number of useful transcriptions with sufficiently high confidence score in the STT output, and therefore improve recall and precision for the approach, there are several factors that will affect the number of related videos being available online, such as: popularity and maturity of a software product, and the purpose of the video being analyzed (e.g., coding related, usage related, design related).

Internal Validity. In this work, we were able to gain high precision and recall values for the proposed mining methodology for crowd-based speech documentation. Nonetheless, various characteristics in the speech data output from STT tools can affect the results. STT tools typically output speech data with only limited punctuation, limiting the applicability of text analysis tools and APIs. Our approach mitigates some of this, by adopting a technique to segment text into sentences based on the pace of the voice in a speech document and pauses between words/sentences. Nonetheless, the resulting sentences are not always well-structured, nor do they map to normal full sentences in written documentation. To further mitigate this problem one could create domain-specific training data for speech text analysis that would allow us to build a language model during the sentence segmentation phase to improve the sentence structure of the transcripts.

Another potential threat to the internal validity of our approach is that the grammar used in spoken text differs significantly from the one in written text. This currently limits the use of written text dependency parsers for identifying sentence boundaries. This threat could be addressed by using spoken text dependency parsers as an alternative solution for sentence splitting.

A threat for the use of STT tools and the quality of produced transcripts is the dialect of the speaker in the speech files. In our dataset, we analyzed videos created by native and non-native English speakers and their English dialects (e.g., British, American, Indian, etc.). For our experiments, we used the US English dialect for analyzing the videos, potentially affecting the quality of our transcripts. Mitigating this thread would require having an STT tool that supports multiple dialects in different languages.

In addition, low speech quality of the screencasts can reduce the quality of the transcripts in terms of their confidence score. We addressed in general the problem of low-quality transcripts by filtering out those screencasts for which the STT tool reported a confidence score of less than 0.7. This filtered out 33 videos (see Table 7-2).

Reproducibility and Reliability. Our study has high reliability because the data is publicly available, we rely on publicly available third party STT tools and information extraction APIs, and our measures are proxies of those used in previous work [87]. Other researchers can replicate and expand upon our results.

7.6 Chapter Summary

This chapter describes an application of how our mining crowd-based screencasts can be applied to enrich and complement existing software project documentation with use-cases extracted from the speech component of screencasts. More specifically, we extract the speech component of YouTube videos, transcribe the speech and apply various Information Extraction techniques to be able to semantically link the speech content to existing project documentation. We presented a proof of concept implementation of our proposed approach and conducted a case study on WordPress tutorial videos posted on YouTube to illustrate that it is possible to extract useful software documentation, in our case, use-case scenarios. We also showed that the use of the analyzed video content can provide an improved ranking of crowd-based documentation resources with a speech component.

In the following chapter we describe another application of our proposed methodology that tries to locate source code artifacts corresponding to the implementation of an application feature that is demonstrated in a screencast.

8 Feature Location Using Crowd-based Screencasts

8.1 Introduction

As discussed in Chapter 1 and further supported by our user survey (Chapter 5), product reviews, how-to videos, tutorials, Q&A sites like Stack Overflow[34] have started to replace many of the more traditional forms of documentation media. As a result, open source users and contributors often have to resort to the Internet for help in finding documentation and explanation to support their current work context. However, with the ever-increasing volume of crowd-based information and resources, the available documents are fragmented across hundreds of textual web documents as well as multimedia documents (e.g., screencasts and podcasts), making it difficult for users to locate and navigate through relevant artifacts (see Chapter 2). One approach to find relevant information is to automatically mine such crowd-based documents and link them with other software artifacts.

In this Chapter, we present an instantiation of our proposed approach (see Chapter 6) that leverages high-level information found in both the audio (i.e., speech) and visual content (i.e., image frames) of screencasts to locate application features presented in the screencasts and link them to their corresponding source code implementations. We conduct a case study on 10 WordPress screencasts that showcase how to use a WordPress feature and 89 Mozilla Firefox screencasts that demonstrate the usage of Mozilla Firefox preferences' features. The results from our case study showed that our approach is capable to locate source code artifacts that are relevant to such screencasts.

In this chapter, we aim to answer the following research questions:

- **RQ7.** What is the performance of the approach using unigram topic modeling vs bigram topic modeling?
- **RQ8.** How accurately can source code files be located from screencasts?

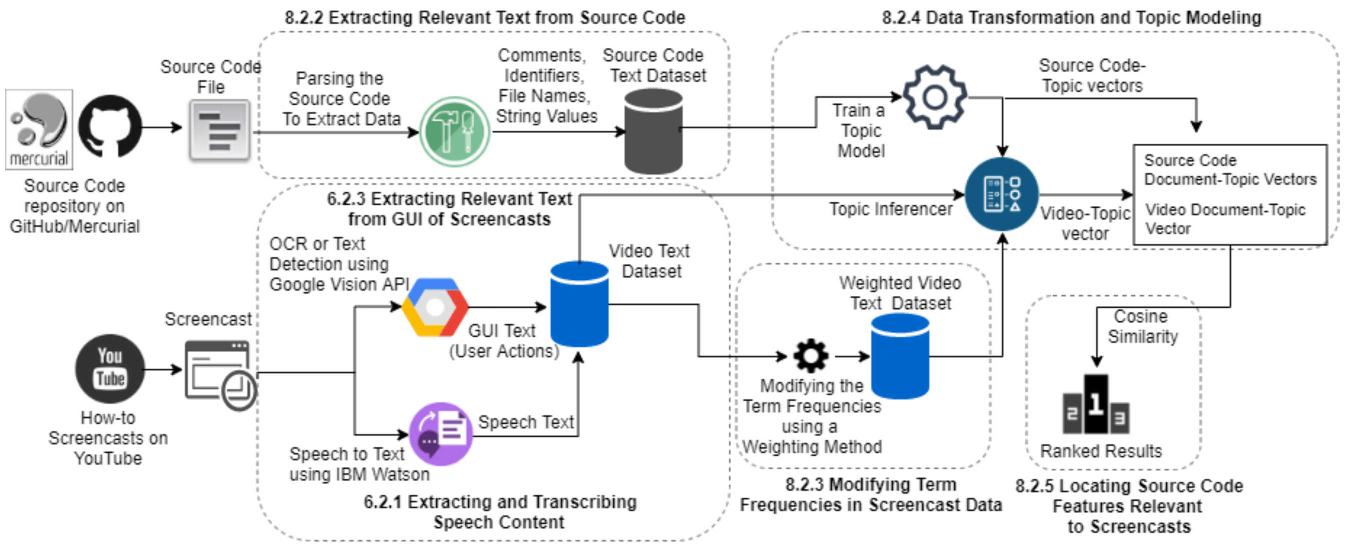
[34] https://stackoverflow.com

- **RQ9.** How does applying our term weighting approach on GUI and/or speech data affect the feature location performance?
- **RQ10.** Are both speech and GUI text data required for feature location?
- **RQ11.** What is the performance of the approach in locating source code related to frontend and backend implementation of a project?
- **RQ12.** How accurately can our approach locate source code directories compared to a guided search approach?

8.2 Methodology

Given a set of screencasts related to a given feature (use-case scenario), our objective is to perform feature location using a screencast and locate the source code artifacts executed by it. Here we leverage or proposed methodology of mining screencasts (Figure 6-1. Methodology). Our proposed methodology, which is detailed in Figure 8-1, is independent of the underlying application or use-case and includes three main steps. We described video text extraction (Sections 6.2.1 and 6.2.3) in Chapter 6. Here we describe first the datasets that are specifically involved in this application of the methodology, which is source code datasets (Section 8.2.1).

Furthermore, the quality of a topic model also depends on the data being used as input. During our initial evaluation of the topic modeling approach, we observed that even after preprocessing, the screencast data contained a significant amount of noise. For a single screencast document containing GUI, speech or both types of data, certain words might be less important to the use-case scenario under study yet occur more frequently than words that are essential to this scenario. Furthermore, some frequent words might occur more often in a single document than compared to the whole data set (i.e., have a low corpus-wide frequency). To address this imbalance, we modify term frequencies within screencast documents based on their importance, using an algorithm that is presented in Section 8.2.3. We then describe our data transformation and topic modeling (Section 8.2.4), and source code feature location approach (Section 8.2.5).


8.2.2 Extracting Relevant Text from Source Code
8.2.4 Data Transformation and Topic Modeling
Source Code File
Parsing the Source Code To Extract Data
Comments, Identifiers, File Names, String Values
Source Code Text Dataset
Train a Topic Model
Source Code-Topic vectors
mercurial
Source Code repository on GitHub/Mercurial
Source Code Document-Topic Vectors
Video Document-Topic Vector
6.2.3 Extracting Relevant Text from GUI of Screencasts
Topic Inferencer
Video-Topic vector
OCR or Text Detection using Google Vision API
Video Text Dataset
Weighted Video Text Dataset
Screencast
GUI Text (User Actions)
Cosine Similarity
How-to Screencasts on YouTube
Speech Text
Modifying the Term Frequencies using a Weighting Method
Speech to Text using IBM Watson
Ranked Results
6.2.1 Extracting and Transcribing Speech Content
8.2.3 Modifying Term Frequencies in Screencast Data
8.2.5 Locating Source Code Features Relevant to Screencasts
Figure 8-1. Methodology for feature location from screen cast interactions to source code.

8.2.1. Data Sources

We use two data sources for our approach: data extracted from screencasts and source code. We described the data sources, extraction and processing of the screencasts in Chapter 6.

Our second data source consists of the source code of the software release exercised in the screencasts. In contrast to the screencast data set, the textual code information is easy to extract from a software's version control system and contains mostly low-level information such as source code text and comments. A main challenge when dealing with such low-level information is to extract and abstract meaningful semantic information, to reduce the semantic gap between the vocabulary used by screencasts and by the source code. To close this semantic gap, different elements of source code information should be used to recover the semantics of the developers' objectives [29], [54], [72], [91], such as:

- ***Source code comments:*** Comments explain or annotate parts of the source code, usually in higher-level terms than the code itself.
- ***Variables and identifiers:*** Variable names usually relate to the scenario-related information stored within them.
- ***String literals:*** Static string literals often appear on GUI widgets.
- ***Method and class names:*** Developers choose class names and method names closer to the software's domain. These names may also map to the features and their relevant text that appear on the GUI of the application.

- ***File names:*** File names are another useful source of information for feature location, since they typically describe the objective or usage of their source code content.

8.2.2 Extracting Relevant Text from Source Code

Source Code Preprocessing: For every source code file, we extract source code elements using Exuberant Ctags[35]. From the full path to a source code file, we extract only the file name (e.g., "browser-customization" from "browser-customization.js"). These source code elements are further processed by removing noise in the data, such as special characters, numbers, punctuations and stop words. For the remaining tokens, we perform identifier splitting using '_', camel case and word splitting (e.g., words that contain special characters, such as '-' in 'menu-item' are replaced by a space), as well as word stemming to further normalize and improve the later linking to relevant code elements.

8.2.3 Modifying Term Frequencies in Screencast Data

We rebalanced the occurrences of terms within screencast documents based on their importance, using a weighting algorithm to modify the term frequencies in a screencast document. The commonly used tf-idf (term frequency-inverse document frequency) [92] approach does not apply in our case. In a preliminary study using tf-idf term weighting on our screencast data, we observed the tf-idf did not improve the performance of our approach. This is because tf-idf reduces the weight of frequent terms in favor of words that are rare across the whole corpus. However, in our case, since we use LDA, which is based on a corpus-wide co-occurrence frequency of the terms, these corpus-wide frequent terms are therefore relevant for locating features.

We instead applied a rebalancing approach that favors screencast documents that contain terms that are more likely to be relevant to source code documents. This rebalancing approach allows us to significantly reduce the effect of noise (i.e., non-relevant terms for a given scenario, OCR and STT errors) in each document, especially when combined with bigram topic models. Our rebalancing approach calculates the term-frequency ranks in a single document based on the corresponding term-frequency ranks in the whole corpus across all available documents of the

[35] http://ctags.sourceforge.net/

same use-case scenario. For the rebalancing, we first merge all screencast text documents, each of which containing the $Diff_{final}$ value (formula 6-2) of a screencast related to the same use-case scenario, into one single document, then calculate term frequencies for this single (all screencasts related to the same scenario) document. Using these term frequencies, we can now determine the relative importance of each term and rank these terms in each single document based on their assigned corpus-wide term frequency. During the last step of our rebalancing, we modify the term frequency of each term j that appears in each screencast document using the following formula:

$$New\_TF_j = round(TF_j \times \frac{1}{New\_r_j}) \qquad (8\text{-}1)$$

Using the $round()$ function, we round off the New_TF_j to its nearest integer, with New_r_j being the new local rank of the term j in a single document based on its corpus-wide rank. Using this approach, the term frequencies in each document will be modified to adjust their weight based on the importance of terms across all screencasts for a given scenario. At the same time, lesser or not important terms will have lower term frequencies or are removed completely from the document (i.e., $New\_TF_j = 0$). For example, given a non-relevant term that is ranked #10 for the whole corpus, and is the most frequent term in a given document (e.g., $TF_j = 35$ and $r_j = 1$). After applying our term weighting approach, its new local rank (New_r_j) in the document will be reduced for the document and accordingly its New_TF_j will be modified proportional to its importance or new local rank (e.g., $New\_r_j = 5$ and $New\_TF_j = 7$). The value of New_r_j or the new local rank of a term, shows how many of the other terms that are ranked above the current term (i.e., their corpus-wide rank is higher) exist in the screencast document. In the above example, the corpus-wide rank of the term was #10 and its new local rank is #5, this means that, 5 out of 10 terms did not exist in the screencast document, while they appear in other screencasts.

The rebalanced screencast documents are also used as input to our topic inferencer, which was created using our source code data set to be converted to screencast document-topic vectors and used as our query vectors.

In what follows we describe our approach of linking screencasts to source code.

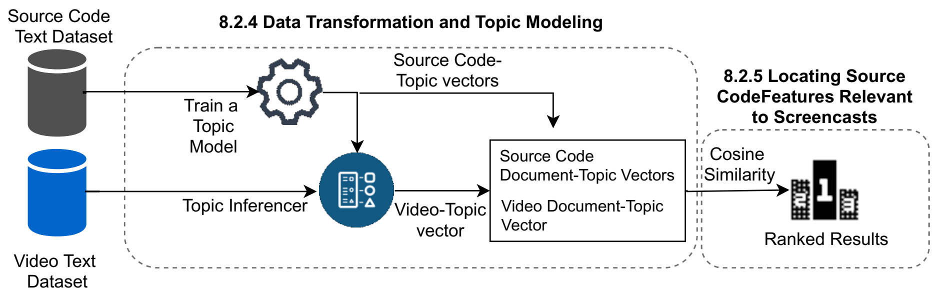
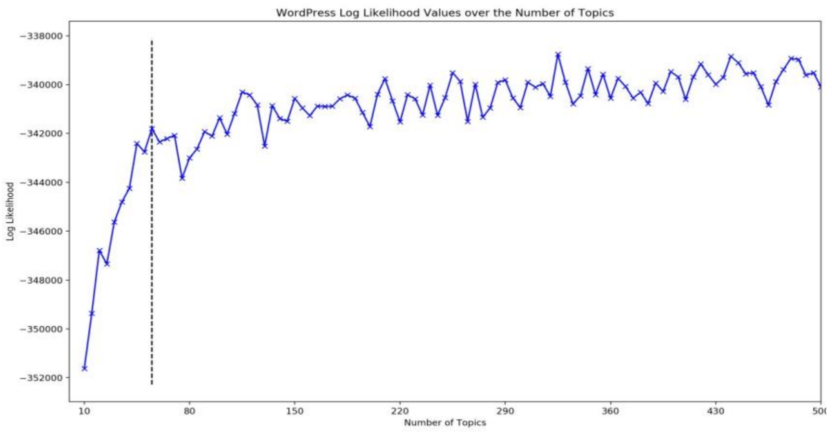

Source Code Text Dataset
8.2.4 Data Transformation and Topic Modeling
Source Code-Topic vectors
Train a Topic Model
Topic Inferencer
Video-Topic vector
Source Code Document-Topic Vectors
Video Document-Topic Vector
Video Text Dataset
8.2.5 Locating Source CodeFeatures Relevant to Screencasts
Cosine Similarity
Ranked Results

Figure 8-2. Feature location using screencasts.

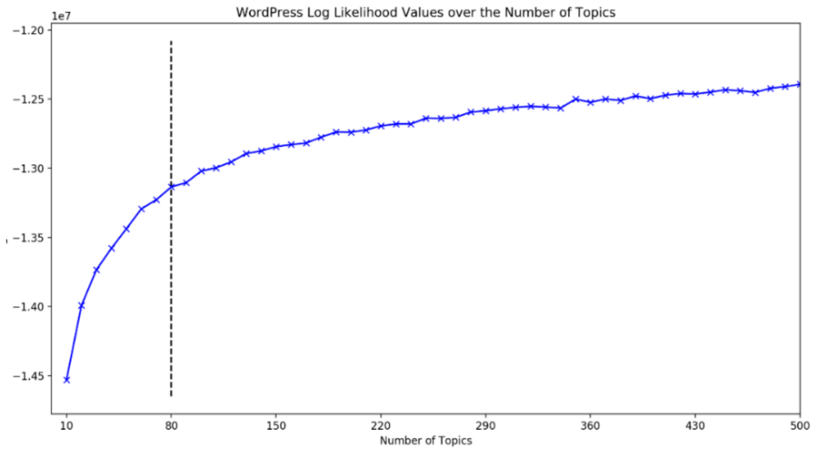
8.2.4 Data Transformation and Topic Modeling

After extracting speech and OCR (GUI) data from screencasts, to extract additional semantic structures from the textual representation of the screencast data and other artifacts such as source code, we use LDA [43] as a topic modeling approach (Figure 8-2). As described in Section 3.2, in LDA each latent topic is characterized by its statistical distribution over a bag of words, and documents are represented as random mixtures over these topics. LDA transforms each document into vectors of topic probabilities, which are then compared with each other. In our case studies (Section 8.3), documents are either source code files or text files containing information extracted from a screencast (GUI, speech or both), with all extracted text being stored in the same document.

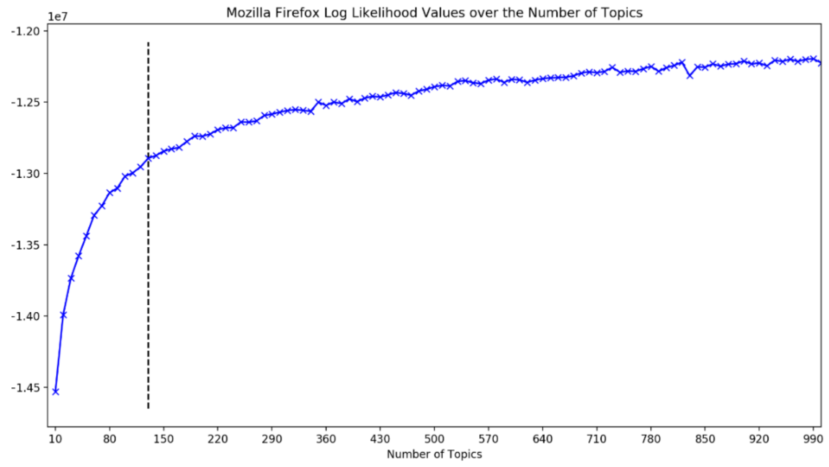
To use LDA, one has first to determine the number of topics that should be used for the topic modeling process. The smaller the corpus, the fewer topics should be generated, while for a larger corpus more topics can be generated [71]. To find the optimal number of topics one can evaluate different configuration settings [93], with different number of topics and manually compared the granularity of the results or evaluate them. Another approach is to perform “knee” (a.k.a., “elbow”) analysis which is used by Thomas et al. [94] who suggest using different numbers of topics and evaluate the resulting models based on their topics’ log likelihood values. The optimal number of topics can be determined by the point where the log likelihood values start to get diminishing returns (i.e., a “knee” in the corresponding plot), which represents a good balance between topic richness and overspecialized topics and avoids having too few or too many topics. In our linking process we used the knee analysis approach. As an example, Figure 8-3 shows the corresponding plots and knees for the two systems used in our case studies in Section 8.3.



a. WordPress – 55 topics using bigram topic modeling.



b. WordPress – 80 topics using unigram topic modeling.



b. Firefox – 130 topics using bigram topic modeling.

Figure 8-3. Log likelihood values vs. Number of Topics (K).

We also can specify the gram size (see Section 3.2) for topic modelling. For instance, bigram topic models split the text into word groups of length 2 (e.g., ‘add_new’, ‘blog_post’, ‘search_engine’, ‘remove_password’). Using bigram topic modeling, the model considers the sequential order of words (after the removal of stop words) by exploiting that word A was immediately succeeded by word B [44]. As a result, words that rarely occur in a corpus (e.g., due to errors and noise from STT and OCR tools) will have very low co-occurrence in bigrams.

ln Jurafsky et al. [95], the authors performed a comparison of statistical language models using different n-gram models. Their study shows that bigram models result in lower perplexity (higher log-likelihood values) and therefore better models compared to unigrams. As Figures 8a and 8b show, when we compared the log-likelihood values between n-gram models for our WordPress data set, our analysis also showed that bigram topic modeling (with vocabulary size of 8,014 words before applying text preprocessing) outperforms unigram topic modeling. Bigram topic models are considered to be more useful for understanding the semantics of a text [96], since the meaning of some words may be affected by their precedent or subsequent word in a sentence (e.g., ‘search’ and ‘engine’ vs. ‘search_engine’). Specifying the gram size can be used to adjust the granularity level of the topics as well as reducing the effect of noise in the text and errors in both OCR and STT outputs.

As Figure 8-3a and 8-3b show, using bigram topic modeling the number of topics for our WordPress source code dataset will be reduced from 80 to 55 topics. For projects, such as Mozilla Firefox (used in our case study in Section 8.3) (Figure 8-3c), where one has to deal with a large vocabulary size (51,294 words before preprocessing the text), multiple programming languages, and different abstraction layers in the implementation, using bigrams can further improve the interpretability of the documents that contain the same terms in different contexts or scenarios.

After determining the number of topics, to perform source code feature location using screencasts as queries we train a topic inferencer using our source code corpus. The topic model creates document-topic vectors from the source code documents. Each element of these vectors corresponds to the probability that a given topic occurs in a document. Next, to be able to use screencast documents to query our source code data set, we use the trained topic inferencer to infer the topics of an input screencast document. We infer screencast documents topics using our source code data set topic inferencer to convert the screencast documents to document-topic vectors that contain the same topics as our source code document-topic vectors.

8.2.5 Locating Source Code Features Relevant to Screencasts

For the last step of our linking process (Figure 6-1. Methodology and Figure 8-2), we use the screencast document-topic vectors as queries and source code document-topic vectors as corpus, to identify source code file(s) that are most relevant to a given screencast. We use the cosine similarity measure [92] to compare the screencast vector with each source code vector. The highest ranked vector corresponds to a file with the highest similarity score, which is therefore the most relevant for a given screencast. Eventually, a list of source code files is obtained which is ranked from most to the least relevant for a given screencast. These ranked files can now be used as a starting point (seed) for further source code navigation during the feature location process.

Since our goal is not to link source code shown in videos to their source code files, it should be noted that our work differs from that of researchers like Khandwala et al. [13] and Ponzanelli et al. [3], [12]. Indeed, we instead propose an approach that can link content (features) demonstrated in screencasts to the corresponding source code implementation of the demonstrated tool, not the source code shown on screen.

8.3 Case Studies

In this section, we evaluate our feature location methodology using two distinct data sets to address our research questions.

8.3.1 Case Study Setup

The goal of our case study is to gain new initial insights on how the linking of GUI features/elements shown in screencasts to the source code artifacts implementing these features might be affected when these source code artifacts either belong to the backend or frontend of an application. Additionally, we also study how the project size (with WordPress being a medium-sized and Mozilla Firefox a large project) may affect our linking results. We therefore measure the accuracy of our feature location approach using both:

1) a project (WordPress) whose backend and frontend components[36] closely interact with each other and are also mostly implemented using a single programming language

2) a project (Firefox) that follows a stricter multilayered architecture, with frontend and backend not only being decoupled from each other but also being implemented using different programming languages[37].

Data Preparation: For our case studies, our target tutorial screencasts are videos in which a narrator explains the use of an application to perform a specific task. For our study, we considered screencasts for two applications, i.e., WordPress and Mozilla Firefox. Both projects differ significantly in their application domain, their size and architectural design (e.g., their front- and backend implementations). Given the popularity of these projects, for each of them many tutorial screencasts have been created and made available on YouTube that explain how to use the features of these projects. Furthermore, at the time of conducting this research, WordPress is ranked as the best CMS tool[38] in terms of providing tools, variety of themes, and affordable cost. It is also known as the most popular one used by more than 60 million websites[39]. Mozilla Firefox is known to be one of the top browsers for its speed and security[40] and is the second most popular browser after Google Chrome[41].

A common challenge when analyzing screencasts is that video quality (resolution) differs among screencasts, which will affect the OCR processing. Since our objective is not to evaluate the quality of the OCR, but rather the quality of our linking approach, we consider only High Definition (HD) videos for our study. If a screencast has a narrator, describing the features being displayed during the screencast, the narration should be in English. In addition, we only considered screencasts for popular features (or scenarios) that are available in the same WordPress or Firefox releases.

In our approach, we train a topic inferencer using the text extracted from source code (Figure 8-2). Then each single video of the same scenario is used to infer its topics (screencast document-

[36] https://codex.wordpress.org/images/2/20/WP_27_modules.JPG
[37] http://tiberius.byethost13.com/pcw_lab/lab1/assign1.pdf?i=1
[38] https://www.techradar.com/news/best-cms-of-2018
[39] https://en.wikipedia.org/wiki/WordPress
[40] https://www.toptenreviews.com/best-internet-browser-software
[41] https://en.wikipedia.org/wiki/Firefox

topic vectors). The resulted screencast document-topic vectors are used as queries and their similarity to source code document-topic vectors are calculated to retrieve and rank the most similar source code documents. As a result, for the purpose of evaluating the results against different baselines (e.g., performance of the approach on frontend vs. backend in Firefox or All vs. Unique in WordPress) we use multiple videos since the number of data points in our evaluations (see boxplots in Section 8.3.2) directly depends on the number of available screencasts (and speech). In addition, the number of available screencasts (and therefore image frames) for each scenario can also affect the term weighting results, since these weights depend on the term frequency of important words).

In what follows, we describe in more detail this screencast selection process and the data preparation steps we applied for the two case study projects (WordPress and Firefox).

WordPress Video Selection and Data Preparation:

Common to CMS tools such as WordPress is that they provide a dashboard that allows users to select functionalities and features for creating content pages. Typical steps to create a website using WordPress[42] include a) choosing and purchasing a domain and web hosting, b) installing WordPress, c) choosing and/or installing a WordPress theme and configuring it, d) publishing a page, e) creating a menu, f) configuring the WordPress settings, and, g) installing WordPress plug-ins. Among these steps, publishing a page (or post) and creating a menu are the main steps than do not require installing third-party tools or plug-ins and can be done through the WordPress default dashboard. Therefore, for our study, we therefore selected two scenarios from the WordPress online documentation that describe such dashboard use: 1. "*How to add menus to WordPress*" and 2. "*How to create a post in WordPress*".

For the selection of the screencast scenarios, we first we performed different queries on YouTube using keywords being selected from the projects' official online documentation[43]. The first scenario describes an administrative task in which a WordPress user has to enter only a limited amount of free-form text, while for the second scenario, the user has to provide a substantial

[42] https://premium.wpmudev.org/blog/a-wordpress-tutorial-for-beginners-create-your-first-site-in-10-steps/
[43] https://wordpress.com/ and https://support.mozilla.org/en-US/products/firefox

amount of text. The amount of user-specific text in these scenarios provides us with variation in the data in terms of different signal-to-noise ratios.

We restricted the selection to screencasts to videos that are in HD quality and were uploaded in the same year as each other to determine the most popular version of WordPress (with most videos available on YouTube), since videos that are uploaded in the same year are more likely to be related to the same version of WordPress. Our analysis shows that the largest number of uploads for both scenarios is in 2015. WordPress version 4.3 is the latest version of WordPress in 2015 and therefore its source code will be used in our analysis. While YouTube provides an advanced filtering feature to limit the length of videos to a maximum length (e.g., less than 4 minutes), such filtering would also return videos that are very short or do not contain any useful content, and hence would again add noise in the data set. We therefore eliminated through manual filtering videos that are too short (less than 120 seconds in our data set) and chose videos that were shorter than 10 minutes, since in our data set videos of longer than this would contain more noise or cover multiple topics. After applying the other selection criteria, the length of videos in our data set is between 120 to 480 seconds. Limiting our data to shorter videos provides us with a more balanced data set that contains less noise, since these shorter videos are more likely to only cover one single usage scenario. For our WordPress case study, we limited our search to screencasts that have both English narration and contain only English text on the screen. For each scenario, 5 screencasts that met our selection criteria and covered the same WordPress version were selected and downloaded using youtube-dl[44] (Table 8-1).

Next, we downloaded the source code for WordPress version 4.3 from the project's GitHub repository[45], and manually replicated and recorded the execution traces of these scenarios using a PHP profiler. We then used Exuberant Ctags to extract source code data (e.g., variables, comments, identifiers - see Section 8.2.1 for more details). Since most of the WordPress features are developed using PHP, we only considered the PHP files for our study. We extracted for WordPress 554 PHP files, with a median size of 162 lines of code with a 25th/75th percentiles of 52 and 511.

Mozilla Firefox Video Selection and Data Preparation:

[44] https://ytdl-org.github.io/youtube-dl/index.html
[45] https://github.com/WordPress/wordpress-develop/tree/4.3

Firefox is among the most widely used multi-platform Internet browsers on the market. Browsing the Internet involves several remote server calls to asynchronously load remote page elements. Since these remote method calls are not part of the Firefox's core JavaScript or C++ application source code, they are difficult to trace and analyze. We therefore focus in our study only on features that involve modifying or executing browser-level functionality that is executed on a user's local machine (e.g., modifying security settings, managing saved user logins, saving web pages).

Firefox main menu is divided into categories and from these categories we excluded features that are related to design (e.g., "Customize" menu) since they are simple scenarios that typically can be done in few steps. We also excluded plug-ins (e.g., "Add-ons") since they do not exist on the default installation of Firefox and therefore are not directly counted as Firefox features. Instead, we investigated the menu items and their sub-menus to locate features that provide more advanced browser-level features and functionalities such as privacy and security, or preferences and options that are related to web browsing functionalities since they are more popular and complicated.

As a result, for our case study, we selected the following seven scenarios from the Firefox online documentation: 1. "*How to import bookmarks*", 2. "*How to change the default search engine*", 3. "*How to set the homepage*", 4. "*How to save a web page to PDF*", 5. "*How to remove saved logins and passwords*", and 6. "*How to clear history and cache.*

Given the long development history of Firefox and the large number of available videos on YouTube covering these Firefox features, we limited our search to screencasts that were uploaded during the last year (at the time of conducting this research) to further improve the ability to locate screencasts that are likely to be related to Firefox Quantum (Version 65), which has been available since the begin of 2019. Similar to the WordPress scenarios, these screencasts contain both information directly related to the Firefox application, but also some user specific (noisy) information (e.g., the scenario "*How to save a web page to PDF*" will typically also show a random web page that will be saved). Like for the WordPress data, we try to reduce the noise and imbalance in the data by limiting the length of the selected screencasts. For the Firefox videos we restricted the video length to 10 minutes since videos of this length mostly cover a single use case scenario and contain less noise. This resulted in a video data set consisting of videos that are between 28 to 430 seconds long.

For our Firefox case study, the goal is to evaluate the performance of our approach on a large-scale project in which backend and frontend implementation of features are developed in different architectural layers that do not interact with each other directly and are developed using different programming languages that follow different coding (e.g., comment, naming, etc.) conventions. To address this challenge, we increased the number of videos to be used as queries for locating source code artifacts. To be able to identify additional videos for the studied release of Firefox, we no longer restrict our YouTube search to videos that have both English narration and English-only text on their screen. Instead, we relied on using bigram topic modeling (see Section 5.3) in addition to modifying term frequencies (see Section 5.4) to reduce the effect of such noise in our data. Selected screencasts were then downloaded using youtube-dl (Table 8-1).

We obtained the Firefox source code used in our study from the project's Mercurial repository[46]. Firefox consists of a large codebase with source code written in different programming languages (e.g., JavaScript, C, C++, Rust, Python, XML, XUL). The Firefox backend is implemented mostly in C/C++, while JavaScript is mostly used for the user interface or frontend development. We performed the experiments on a Mac running OS X, which only uses Firefox code that is developed specifically for OS X platforms. Using again Exuberant Ctags, we extracted the source code facts from JavaScript and C/C++ source code files. The analyzed codebase consisted of 36,666 JavaScript files and 33,440 C/C++ files, with a median size of 45 lines of code for the JavaScript files (25th/75th percentiles of 27 and 83) and a median size of 133 lines of code for C/C++ files (25th/75th percentiles of 64 and 332).

Applying our Methodology: For our case studies, we consider the following screencast data: user actions, text shown in the screencasts and narrator speech (if available). For screencasts with a narrator, we used IBM Watson's speech-to-text service (STT) [18] to automatically transcribe the audio part of these screencasts. The tool also provides a confidence score that specifies how accurately each term is transcribed from the speech. To reduce the effect of noise and errors in the speech data, in our WordPress data set, we filtered out words from the transcribed text with a low confidence score (below 0.7). For our Firefox case study, we instead use bigram topic modeling (see Sections 3.2 and 8.2.4), which already mitigates the problem of having errors in our STT

[46] https://hg.mozilla.org/mozilla-central

output by emphasizing the word context, therefore we did not apply a confidence score threshold to the Firefox STT output.

For processing the content of the image frames, we extracted image frames at a rate of one frame per second using FFmpeg[47] (Table 8-1). It should be noted that the number of reported frames (Table 8-1) corresponds to the frames after manually removing the begin/end of each video since these parts typically do not contain any information relevant to the actual scenario (e.g., greetings, information related to the creator or YouTube channel, thank-you notes, or closing remarks inviting viewers to like or share the screencast).

Table 8-1. Number of transcribed speech documents and image frames extracted from each video.

Project	**Usage Scenario**	**Number of Screencasts**	**Number of Speech Documents**	**Number of Image Frames used for OCR**
WordPress	Menu	5	5	1,561
	Post	5	5	749
Mozilla Firefox	Import Bookmarks	12	5	1,349
	Clear History	21	6	1,402
	Set Homepage	21	7	1,393
	Remove Passwords	16	7	1,131
	Save-to-pdf	6	1	680
	Default Search Engine	13	6	694

Next, we apply OCR to extract the textual content of each sampled video frame. We first explored the Tesseract tool[48] used by Ponzanelli et al. in [3]. However, the tool requires a significant amount of pre-processing to improve first the quality of the image frames, by removing the image background, resizing the image and removing frame areas that do not contain any text. We instead opted for Google Vision API's Text Recognition service[49], which does not require these preprocessing steps and can also recognize text on images with any background, while providing the (x, y) coordinates of the area in which the text appears on the image frame as well as a confidence score for the recognized words.

[47] https://www.ffmpeg.org/
[48] https://github.com/tesseract-ocr/tesseract
[49] https://cloud.google.com/vision/

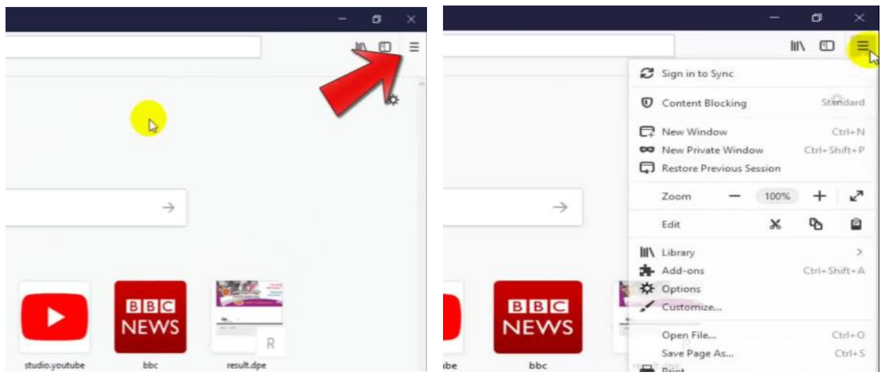


Figure 8-4. A sample user action in a screencast.

Google Vision API is an image analysis service that is built on top of machine learning models that are trained on a large volume of image data with various characteristics (e.g., image background, quality, text font, language, etc.). For evaluating the performance of our user action detection approach (Formula 6-2, Figure 8-4) using Google Vision API, we calculated the precision and recall of the user actions that are identified using our approach. We manually verified, using one randomly selected video, the user actions leading to a change on the screen (e.g., clicks on a button or menu items, scroll up/downs, zoom in/outs) and the corresponding text changes to identify the true positives and false positives.

From our analysis of the raw OCRed text (without any preprocessing), we observed that our approach is able to detect all user actions and text fragments (recall 100%) with a precision of 55%. This somewhat low precision is due to errors in the OCRed text extracted from the image frames. These OCR errors caused 106 out of 324 image frames (32%) to be wrongly flagged as user actions. These false positives mainly occur when two subsequent image frames with very similar content, contain erroneous OCRed words that result in wrongly identifying a new user action due to partial word differences. However, when we compare the manually extracted user actions to the ones extracted by our proposed approach, the length of the texts (i.e., number of words) in the OCRed text that has been identified incorrectly as a user action are short (Median = 5 words). Such false positives (Table 8-2) can be removed by applying additional preprocessing on the extracted text (i.e., modifying term frequencies and bigram topic modeling). In addition, since for each screencast we merge the text of all identified user actions in one large data set, these relatively few short text outliers will become less important. Table 8-2 provides results of our statistical analysis of the word length in each OCRed text based on the text differencing approach introduced in Formula 6-2.

In addition, by using our frame sampling approach (with one frame per second), we can reduce the processing cost for image and OCR processing significantly compared to an approach that would process the full frame rate of videos (e.g., 30 frames per second). Also, our approach is scalable since the OCR (text detection) process is executed in a cloud environment, which allow us to take advantage of elasticity and scalability provided by cloud computing.

Table 8-2. Statistical analysis of the OCR differencing approach on a randomly selected video.

Observation	Minimum Length	Maximum Length	Mean	Median
True Positive	1	86	21.8	19
False Positive	1	63	8.4	5

For our topic modeling we used MALLET[50] to perform LDA. To determine the optimal number of topics for our study, we split the source code data set into a 90% training and a 10% test portion. For the WordPress data set, we relied on a unigram topic modeling, whereas for the Firefox data set, we used bigram topic modeling (see Section 8.2.4). To determine the number of topics, we evaluated both models based on their log likelihood values (Figure 8-3) to identify the "knee" point K where one gets diminishing returns in log likelihood as the number of topics increases. For WordPress bigram topics we identified $K = 55$, for WordPress unigram topics $K = 80$, and for Firefox (bigram topics) $K = 130$, which we used later for our case studies.

Baselines: We evaluate our ranking results against baselines that we manually created by executing the locally compiled and built versions of the projects and recording their execution traces for the scenarios described in the screencasts.

WordPress Baseline Creation: Using Xdebug[51], which is a popular PHP profiler, we create execution traces in the form of a call-graph for each use-case scenario of WordPress. We then parsed the call-graph trace to extract the path for each PHP file whose method executions was recorded.

We create two different types of baselines. The first baseline, the Unique baseline, contains only files whose methods were executed and that are unique to a given scenario. This baseline can be considered to be more relevant to the technical implementation of the essential steps of a particular scenario while still containing a low amount of generic information. In contrast, the All baseline

[50] http://mallet.cs.umass.edu/
[51] https://xdebug.org/

consists of the Unique baseline and in addition also includes methods which might be shared with other scenarios. For example, in WordPress functions.php[52] includes methods that are called during the execution of different WordPress features, since these methods are responsible on how a site is publicly displayed. The *All* baseline is more generic, since it includes both some general executions and executions specific to a given scenario.

Firefox Baseline Creation: The execution traces for the Firefox use-case scenarios are created using its built-in Gecko profiler[53]. The input to the profiler is created by replicating and recording the execution traces for the scenarios shown by the screencasts. The Gecko profiler is a sampling profiler that interrupts the threads that it is profiling at regular intervals (e.g., 1-2 milliseconds by default depending on the platform) and captures the call stack each time the thread is interrupted. As a result, some calls might be omitted in the call-graph since the profiler does not include them in the snapshots of that execution. To mitigate this threat, we profiled the same scenario several times, creating different trace snapshots, which we then combine in a single trace.

The output of the Gecko profiler contains the fully qualified names of the executed methods for C/C++ or Chrome[54] URLs[55] that are used to reference the JavaScript source code files. The mapping of the Chrome URLs to concrete file names is based on manifest files that contain the rules for resolving the URL into their concrete file paths. Instead of decoding the complex rules to translate Chrome URLs to concrete files we used a simpler approach that uses the JavaScript file name in the URL, method name, and the method's line number, which are provided by the profiler, to locate the corresponding file. Also, in Firefox many of the recorded execution traces are lower-level system calls that are part of the general Firefox initialization. These system calls introduce additional noise in the trace data in terms of being not directly related to the specific scenario. To mitigate this issue, we recorded the execution trace of Firefox while it was in idle mode and subtracted this recorded (idle) trace from that of each scenario.

Evaluation Measures: Since the objective of our feature location approach is to identify a starting point (seed) in the source code to be used for further manual feature exploration, we use

[52] https://codex.wordpress.org/Functions_File_Explained
[53] https://perf-html.io/
[54] https://developer.mozilla.org/en-US/docs/Glossary/Chrome
[55] https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/Tutorial/The_Chrome_URL#The_Chrome_URL

Average Precision (AP) and Reciprocal Rank (RR) as evaluation measures to assess the ability of our approach in retrieving true positives at the top of the result set [97]:

$$AP = \frac{1}{|R|} \sum_{k \in R}^{n} Precision\ at\ k \qquad (8\text{-}2)$$

$$RR = \frac{1}{rank\ of\ the\ first\ tp} \qquad (8\text{-}3)$$

with R being the set of all relevant retrieved items and k being the number of retrieved items. We use AP, since it considers the position of the true positives in the ranking and prefer AP over the "Precision at k" measure, because AP is generalizable over multiple queries. Using AP, one can calculate now the mean of the sum of "Precision at k" values for all queries (here we used the median value to avoid bias caused by outlier values). "Precision at k" has this limitation that to have a fair assessment, using for example "Precision at 1000", at least 1000 (i.e., k) actual relevant items must exist in the corpus for all executed queries [97].

RR can be considered as a complementary measure to AP in assessing the quality of our query results. RR considers the position of the first true positive in the search results and is best applied when one has very few true positives (such as in feature location), some of which are located in the top k of the search results. RR therefore evaluates whether the most relevant hits appear at the top of the result set, while AP determines if an approach can retrieve the relevant results and rank them among top hits.

For Firefox, we used directories as our granularity level for the search results, since our initial evaluation at both file and directory granularity levels showed that our approach performs better in locating source code at directory level compared to the file level (our evaluation data can be found online for file56 and directory57 granularity levels). Our analysis for locating and ranking source code relevant to the backend development of Firefox at the file granularity level resulted in median values close to zero for both AP and RR, making our approach not applicable at this granularity level.

We further analyzed the difference between the performance of a file-based approach and a directory-based approach. For this comparison, we calculated the overlap [98] between the

[56] https://1drv.ms/u/s!ArPyXNcsMbKOg-Z5bV4YJ_NMCIFVjA?e=ieKnT5
[57] https://1drv.ms/u/s!ArPyXNcsMbKOg-cAhRQIlVEtzWMyIA?e=THaDl6

vocabulary of our screencast documents (for both Firefox and WordPress) and the vocabulary extracted from the source code files that exist in the baselines that are created for all selected scenarios. Our analysis showed that the overlap between the vocabulary of screencast documents and the corresponding source code in WordPress is 80%, in Firefox for the JavaScript source code it is 43% and for the C/C++ source code the overlap is only 17%. This large difference in vocabulary between screencast documents and the Firefox source code, especially for C/C++ source code, is the main reason why a file-based approach does not perform well for Firefox.

By applying our approach at directory level granularity, we were able to locate and rank in most cases the directories that contain the relevant files. While this approach no longer directly locates relevant individual files, it still allows for a reduction of the search space and provides users with an approach to partially automate the feature location process. Using directory level granularity, a user only has to search manually through an individual (relevant) directory for the source code file(s) implementing the particular feature rather than manually searching across all directories. These ranked directories of files can therefore be used as a starting point (seed) for further source code analysis during the feature location process. Table 8-3 presents statistics about the number of JavaScript/C/C++ files in directories of Firefox source code containing such files.

Table 8-3. Mean, median, maximum, and minimum number of JavaScript and C/C++ files.

Language	Mean	Median	Max	Min	#Directories
C/C++	14.29	3	2014	1	2623
JavaScript	14.91	3	1660	1	4268

For the evaluation, we use the top 10 hits (files) in WordPress and top 100 hits (directories) in Mozilla Firefox. Our initial analysis showed that our approach would only yield no or very few true positives in the top 10 results using the chosen granularity levels. Instead, we used top X/100 as an evaluation measure for the Mozilla Firefox data set, since Firefox is not only a significantly larger project than WordPress, and its front/backend implementation are not only in distant architectural layers (compared to the WordPress implementation), but it also relies on different programming languages. Furthermore, since the maximum number of directories in our baseline for Firefox may contain fewer than 100 directories in the complete result sets for the baselines, we therefore consider top X/100 [97], with X<=100 and X being determined based on the number of directories in the baselines that we extracted for Firefox. For example, if the baseline contains only 60 directories, then the top 60 results are evaluated.

We also calculated random guess probabilities for both WordPress and Firefox, which we use as a baseline for our result comparison. The random guess probabilities were calculated by dividing the number of files/directories in the baseline by the number of all files/directories in the project. For the random guess probabilities, we calculate how probable it is that a developer by randomly (without considering any bias or prior knowledge) selecting files/directories, would select all the files/directories that are captured in the scenario baseline (Table 8-4).

Table 8-4. Random guess probability of the baseline files out of the total corpus of 554 source code files in WordPress, 36,666 JavaScript files, 33,440 C/C++ files, and 7,433 directories containing JavaScript or C/C++ files in Firefox

Project	**Use-case Scenario**	**Granularity**	**Baseline**	**#files or #dirs**	**Random Guess %**
WordPress	Post	file	All files	92	17
			Unique files	14	3
	Menu		All files	92	17
			Unique files	13	2
Firefox	Import Bookmarks	directory	C/C++	56	0.7
			JavaScript	2315	31
		file	C/C++	156	0.5
			JavaScript	1309	4
	Clear History	directory	C/C++	215	3
			JavaScript	1294	17
		file	C/C++	1298	4
			JavaScript	1420	4
	Set Homepage	directory	C/C++	27	0.3
			JavaScript	1288	17
		file	C/C++	66	0.2
			JavaScript	1447	4
	Remove Passwords	directory	C/C++	262	3
			JavaScript	1299	17
		file	C/C++	1898	6
			JavaScript	1470	4
	Save-to-pdf	directory	C/C++	25	0.3
			JavaScript	1131	15
		file	C/C++	63	0.2

		JavaScript	1183	3
Default Search Engine	directory	C/C++	30	0.4
		JavaScript	1331	18
	file	C/C++	91	0.3
		JavaScript	1573	4

8.3.2 Case Study Results

RQ7. What is the performance of our approach using unigram topic modeling vs. bigram topic modeling?

As we use LDA in our linking methodology, different factors can affect its performance, including the number of topics and the gram size. First, we applied unigram topic modeling on our WordPress data set (Figures 8-5 and 8-6). We then extend our analysis using unigram topic modeling to also include bigrams (Figures 8-7 and 8-8) and compare the performance of both gram sizes. As part of bigram topic modeling, the number of topics was reduced from K = 80 (unigram) to K = 55 (bigram), with few topics needed to form a particular well-defined topic context.

Results for "menu" Scenario: Figure 8-5 shows that using unigram topic modeling for the *All* baseline, except for the speech data, the RR values are always 100%, meaning that the first true positive is ranked at the top of the result set. Using bigram topic modeling (Figure 8-7), we have the same results and the median RR value of speech data not only is improved by 75% but also the variance in the distribution of the RR values is reduced. In case of the *Unique* baseline, the RR value of the speech data improved by 30%. This is while the median RR values when using GUI or GUI-and-speech data decreased to 50% and the variance when using GUI and speech data is removed.

The median AP values of bigram topic modeling when using the *All* baseline are all improved compared to unigram topic modeling. Especially when using speech data, the median AP improves by 70% and the variance in the boxplot is reduced. Evaluations against the *Unique* baseline shows that for the speech data the median AP values improved by 30%, while the variance in other boxplots is decreased and their median AP values are also decreased on average by 19% using bigram topic modeling.

Results for "post" Scenario: The evaluations against the *All* baseline (Figures 8-7 and 8-8) shows that the rank of the first true positive drops from the first to the second hit when using GUI weighted or GUI-and-speech weighted data in bigram topic modeling. However, using bigram topic modeling improves the median RR of speech data by 89% and reduces the variations in the box plots. In case of the *Unique* baseline, although the median RR values decreased for five out of six data types using bigram topic modeling, this value increased for the speech data from 0 to 12% and the variations are either removed or reduced which makes the results more reliable.

Evaluations against the *All* baseline using bigram topic modeling (Figure 8-8) show that the AP values of the GUI weighted, and GUI-and-speech weighted dropped from 100% (using unigram topic modeling) to 57%. This means that still the number of true positives is higher than the number of false positives in the top 10 results, and the RR values show that the first true positives appear at the second rank of the top hits. For the other data types the variance in the box plots reduced and the median AP values increased especially for the speech data the increase is by 42%. The evaluations against the Unique baseline show that the median AP values for all data types except for the speech data dropped on average by 9.4% while the variance in the boxplots is reduced and the median AP of the speech data increased from 0 to 17%.

Conclusion: While the improvement in median AP and RR values varies between using bigram vs. unigram topic modeling, in general, using bigram topic modeling reduced the variance in the boxplots, which makes the results more reliable. Also, in all baselines, and for both AP and RR, bigram topic modeling improved the performance of the approach when using speech data, which compared to GUI data is shorter. As a result, we use bigram topic modeling in the rest of this chapter.

RQ8. How accurately can source code files be located from screencasts?

In what follows, we report on the AP and RR results that we obtained when we take advantage of both GUI and speech data for the linking of screencast content to source code (Figures Figure 8-7,Figure 8-8Figure 8-11**Error! Reference source not found.** the far-left blue boxplots).

Results for WordPress:

Results for "menu" Scenario: Figure 8-7**Error! Reference source not found.** shows that the median AP for the *All* baseline is 0.99. Having an AP above 0.5 indicates that we have more true positives than false positives in the top 10 hits, which can be considered a good result for average precision.

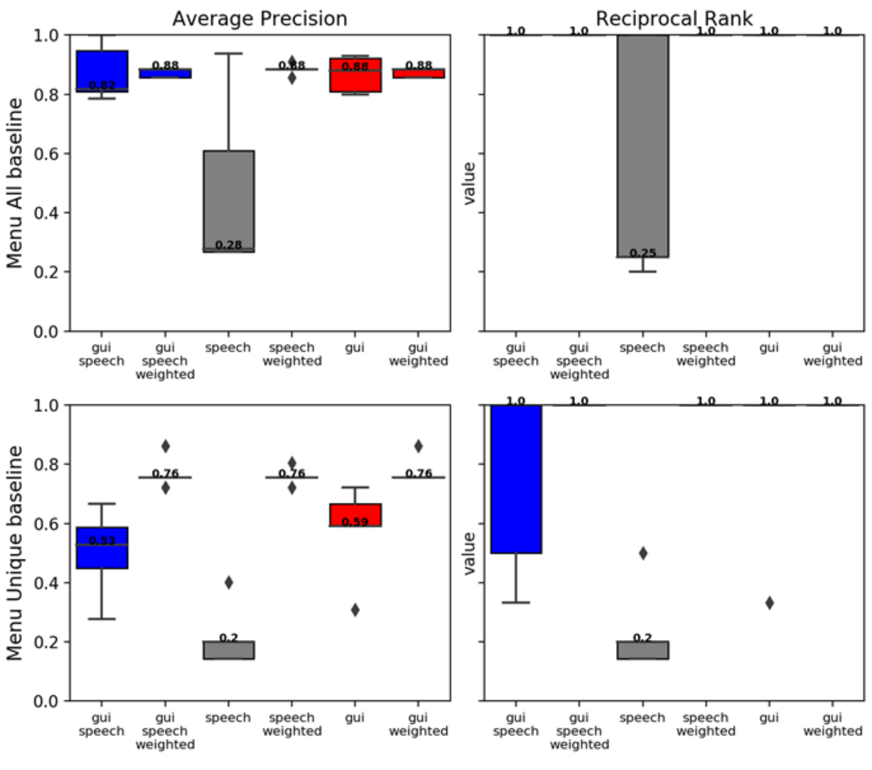


Figure 8-5. RR and AP boxplots for the unigram topic modeling of the "Menu" scenario evaluated against All and Unique execution trace files, using both GUI/speech data, only GUI or speech, either with weighted or unweighted metrics

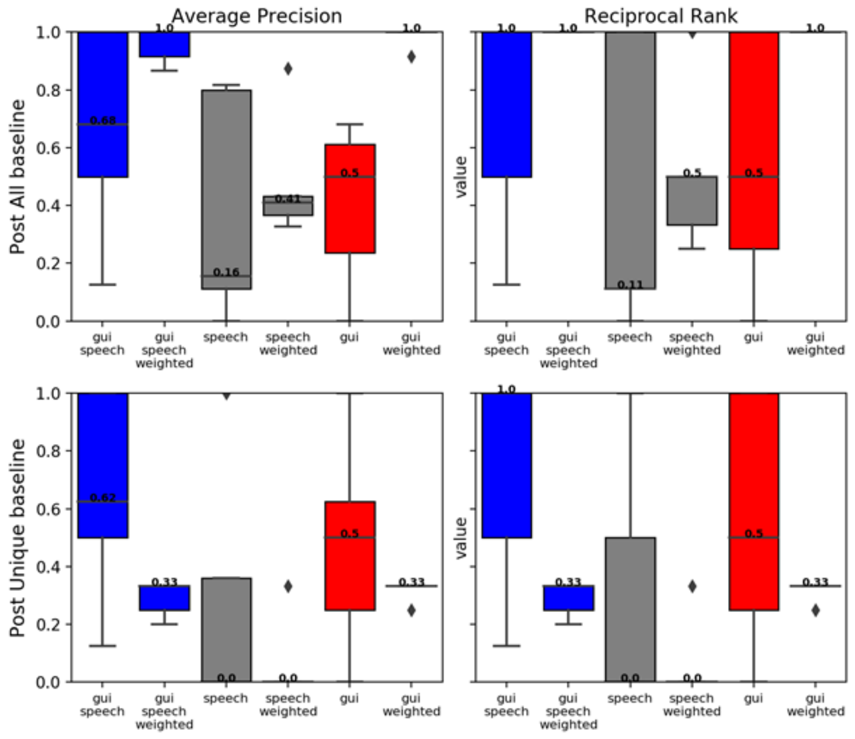


Figure 8-6. RR and AP boxplots for the unigram topic modeling of the "Post" scenario evaluated against All and Unique execution trace files, using both GUI/speech data only, GUI or speech, either with weighted or unweighted metrics

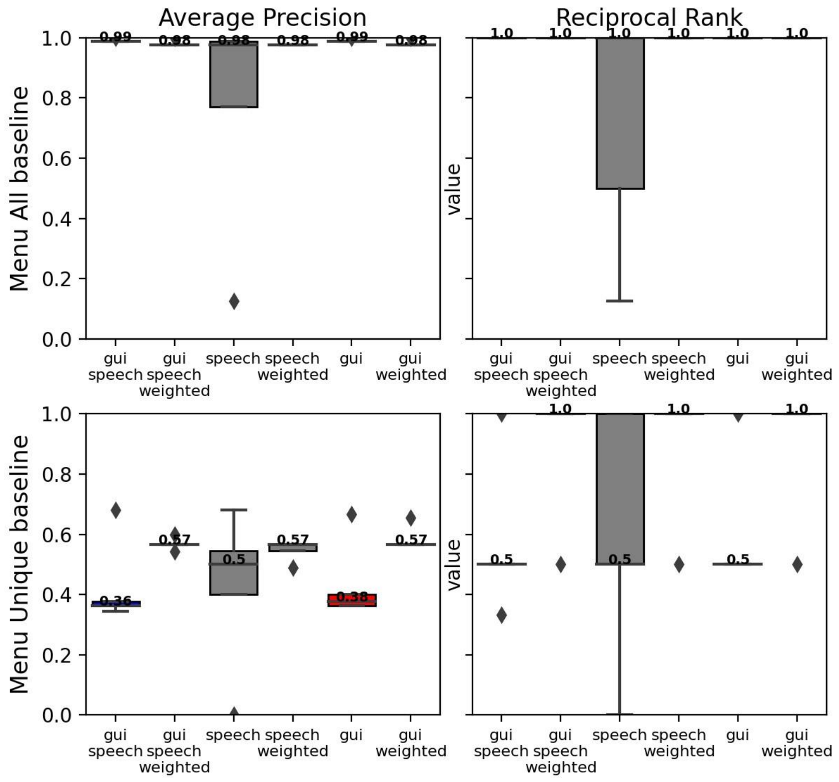


Figure 8-7. RR and AP boxplots for the bigram topic modeling of the "Menu" scenario evaluated against All and Unique execution trace files, using both GUI/speech data, only GUI or speech, either with weighted or unweighted metrics.

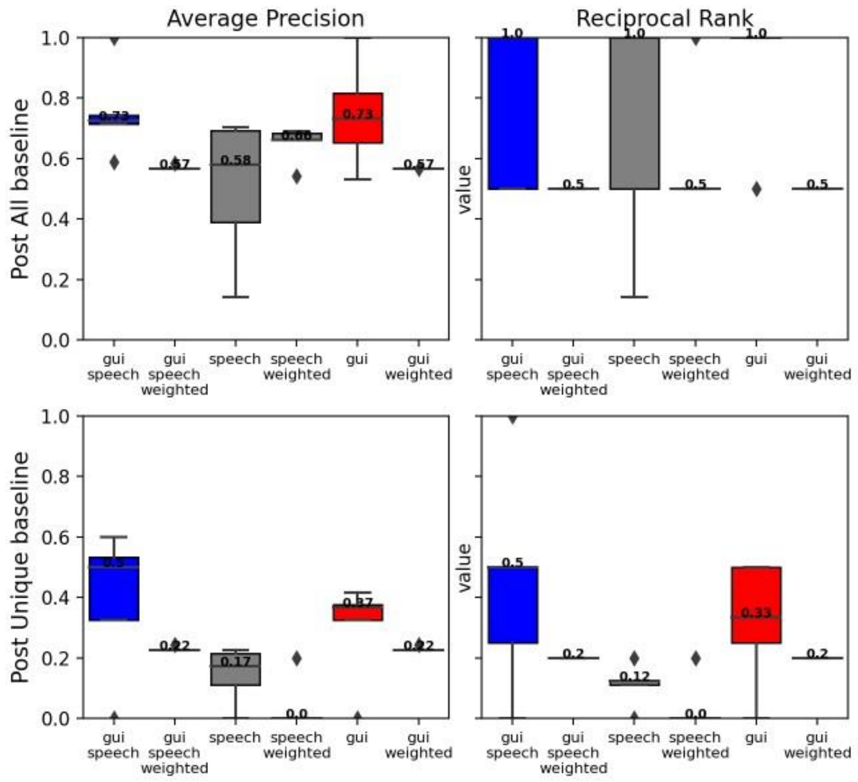


Figure 8-8. RR and AP boxplots for the bigram topic modeling of the "Post" scenario evaluated against All and Unique execution trace files, using both GUI/speech data only, GUI or speech, either with weighted or unweighted metrics.

Given that there are among the 554 WordPress files only a total of 13 PHP files (true positives) for the *Unique* baseline, while the median RR value is 0.5, meaning that the first true positives appears at the second top of the result set, the median AP is 0.36 which means that every third item in the result set is a true positive.

More importantly, all screencasts had an RR value of 100% for the *All* baseline, and the RR values for the *Unique* baseline were 50%. With the first true positive being ranked first or second indicates that our approach can provide useful seeds for further feature location.

Results for "post" Scenario: Figure 8-8 shows that the median of RR for the *All* baseline is 100% and for the *Unique* baseline this value is 50%, which indicates that even for a noisier data set (e.g., "post" scenario), our approach is able to rank in most cases the relevant result in first or second position. Also, the median of AP is 0.73 for the *All* baseline and 0.5 for the *Unique* baseline, which shows that, in most cases, the number of true positives that our approach will return is more than (*All* baseline) or equal to (*Unique* baseline) the number of false positives in the top 10 hits.

Further evaluations for both baselines (Table 8-4) show that the approach always outperforms random guessing by a factor of at least 16.66 for the *Unique* baselines and 4.29 for the *All* baselines. Random guessing refers here to the process where one would try to correctly guess all relevant files in each baseline.

***Results for Mozilla Firefox*:**

We present the detailed AP and RR results for the two scenarios of "*Import Bookmarks*" and "*Default Search Engine*" where the latter belongs to the group of scenarios with more noise in the data (i.e., "*Set Homepage*", "*Save-to-pdf*", "*Default Search Engine*"). The results are shown again in the far-left blue box plot of Figures Figure 8-11**Error! Reference source not found.**. Also, a summary of the median AP and RR values for all scenarios is shown in Figure 8-9. The complete set of all box plots for the Mozilla Firefox scenarios is available online[58].

Results for "Default Search Engine" Scenario: Figure 8-10 shows that also for this scenario JavaScript results outperform our C/C++ results. The first true positive for the JavaScript files appears at rank 5 and for the C/C++ files the first true positive appears at rank 7 of the top hits.

[58] https://1drv.ms/u/s!ArPyXNcsMbKOg7tGilFda7q6Szl61g?e=CsMkK8

Results for "Import Bookmarks" Scenario: Figure 8-11 shows that the median of AP is 0.18 for the C/C++ files and 0.23 for the JavaScript files. This reflects that every fifth result is a true positive among the top hits. The RR values for both C/C++ and JavaScript are the same, with 0.2. For the JavaScript results 25% of the screencasts have an RR value of more than 0.5, which means that the first true positive appears at rank 1 or 2.

Our study (Figure 8-9) shows that, scenarios that are more likely to contain noise or irrelevant data in their GUI text (i.e., "*Save-to-pdf*", "*Default Search Engine*", and "*Set Homepage*") have the lowest median AP and RR values for the C/C++ evaluations. We also observed that noise in the data has a negative effect on both JavaScript and C/C++, however the RR values for the JavaScript (frontend) are less affected than the RR values for the C/C++ files.

In our study, the median AP for both JavaScript and C/C++ always outperform random guessing (Table 8-4), except for the random guess value of the JavaScript directories of the "*Import Bookmarks*". This is because the speech data in this scenario is of very low quality, due to errors in the output of the STT tool (caused by the accent of the narrators) and the significant amount of noise in the data, since the narrators not only talk about importing bookmarks from Firefox but also how to export bookmarks from another browser. As a result, after preprocessing the speech text these documents become very short which negatively affects the LDA performance.

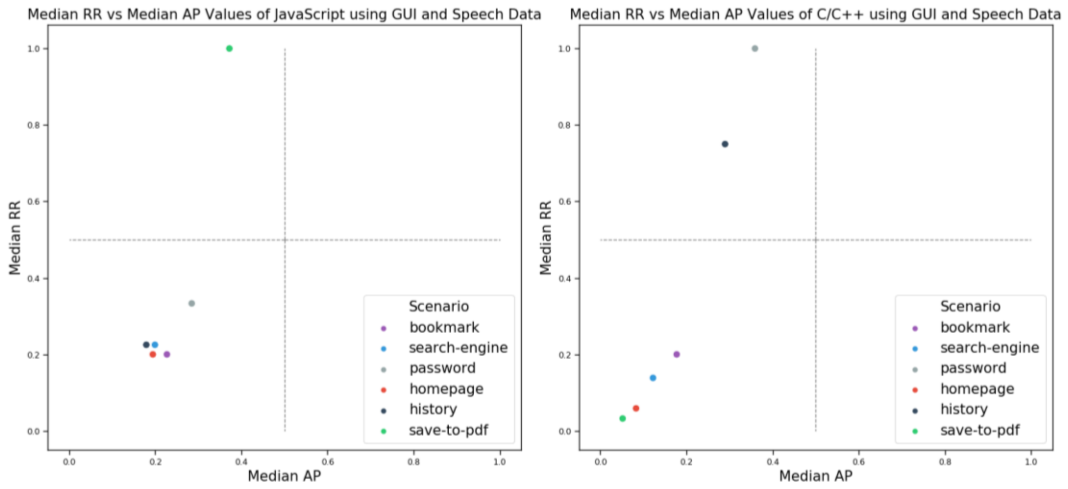


Figure 8-9. Scatter plot of the median AP and RR values for each scenario and each language in Mozilla Firefox using GUI and speech data.

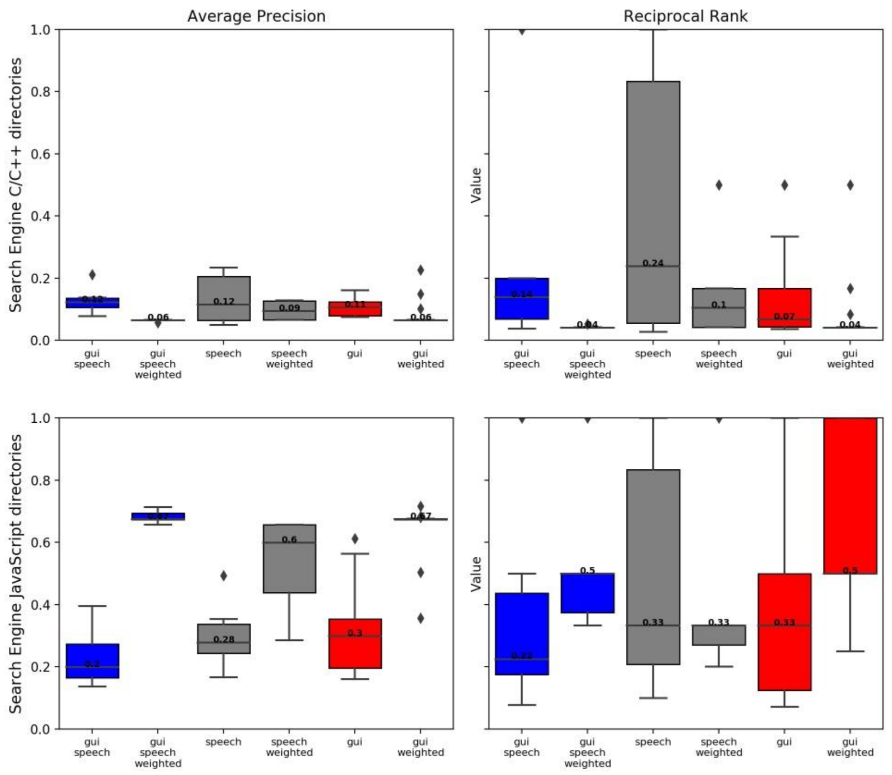


Figure 8-10. Boxplots of RR and AP for the “Default Search Engine” scenario evaluated against C/C++ and JavaScript directories, using both GUI/speech data, only GUI or speech either with weighted or original metrics.

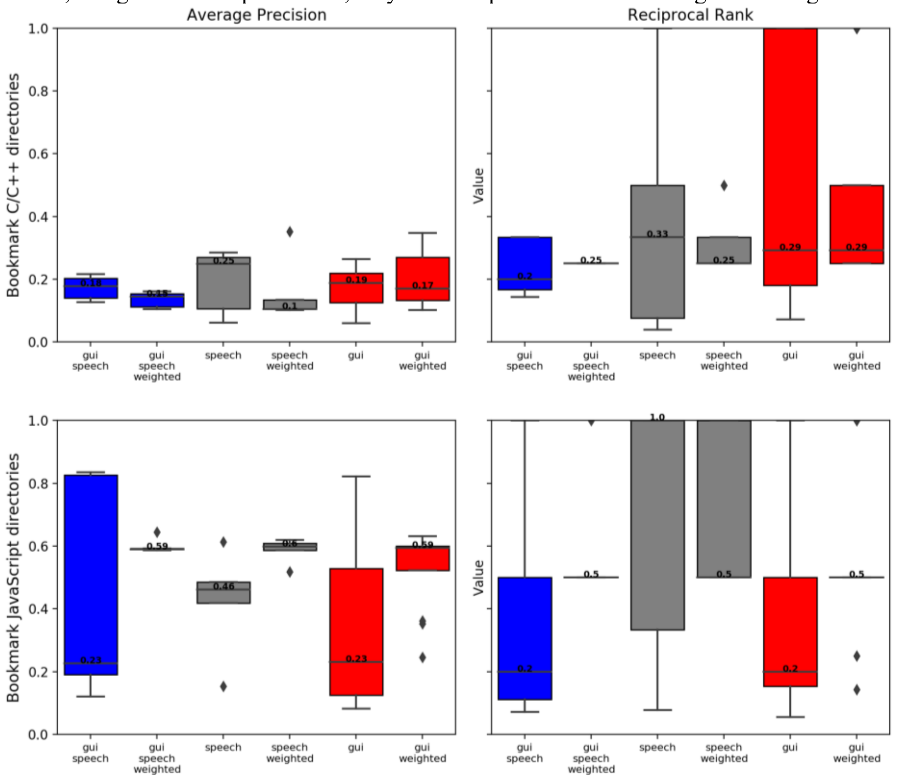


Figure 8-11. Boxplots of RR and AP for the “Import Bookmarks” scenario evaluated against C/C++ and JavaScript execution trace directories, using both GUI/speech data, only GUI or speech either with weighted or original metrics.

Conclusion: In WordPress, our approach can successfully rank the first relevant item at the top of the result set when using the GUI and speech data set, with the results for the "*menu*" scenario being more precise since there is less noise in this data set, which leads to have more terms in common with the source code data set. Even for noisier scenarios, our approach outperforms random guessing (Table 8-4), by being able to retrieve more than 50% of the true positives in the top 10 results. In Mozilla Firefox, with its larger codebase, it is more challenging to obtain accurate results. However, the median AP and RR values of all scenarios show that the approach can return the first true positives in rank 1 to 30 (in the worst case) of the top *X*/100 results. Also, except for the median AP of the JavaScript results of the "*Import Bookmark*" scenario, in all other scenarios, the approach outperforms random guessing.

RQ9. How does applying our term weighting approach on GUI and/or speech data affect the feature location performance?

For this research question, we analyze and compare non-weighted and weighted results (boxplots) in the Figures Figure 8-7,Figure 8-8Figure 8-11**Error! Reference source not found.** ("gui speech" and "gui speech weighted") to gain insights on how adjusting the term frequencies in the GUI/speech data (Section 8.2.3) affects our linking approach.

Results for WordPress:

Results for "menu" Scenario: The blue boxplots ("gui speech" and "gui speech weighted") in **Error! Reference source not found.**Figure 8-7 show that, with the term frequencies adjusted, the median RR for both *All* and *Unique* baselines is always 100%, which means that our approach ranks the relevant results at the top of the result set. Also, for the *Unique* baseline, all RR values are 100%, showing that rebalancing the combination of GUI and speech data improves the RR.

When comparing the AP values (non-weighted vs. weighted) of the *All* baseline shows that in this case applying term weighting does not noticeably improve the performance of our approach. However, for the *Unique* data, term weighting improves the median AP by around 20%.

For the "speech" and "speech weighted" data sets in Figure 8-7, both the *All* and *Unique* baselines applying term weighting on the speech data reduce the variance in the results and significantly improved the median values (on average by ~68%).

For the “gui” and “gui weighted” data sets (red box plots in Figure 8-7), the RR values for both *All* and *Unique* baselines are the same. While the median AP value for the *All* baseline does not change, applying term weighting reduces the variation in the boxplots. For the *Unique* baseline, term weighting improves the median AP value by 17% and removes the variation in the boxplots.

Results for “post” Scenario: **:** The blue boxplots (“gui speech” and “gui speech weighted”) of the *All* baseline (Figure 8-8) show that, modifying the term frequencies again increases both the median AP and RR value to 100% for all screencasts. However, for the *Unique* baseline we observed a decrease for both the median AP and RR. A more detailed analysis of this somewhat unexpected result shows that it is caused by the weighted documents being shorter after the removal of less important words (noise) with a low corpus-wide frequency, and hence leading to a lower TF_j (the occurrences of words). Also, the *Unique* baseline contains fewer files and therefore fewer TP to be ranked compared to the *All* baseline.

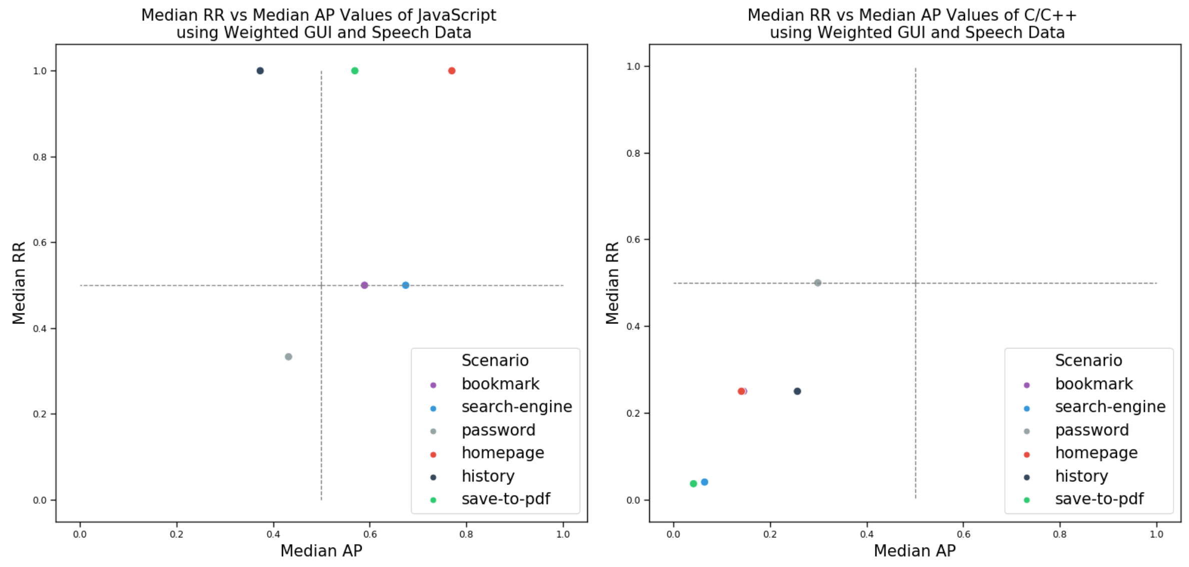


Figure 8-12. Scatter plot of the median AP and RR values for each scenario and each language in Mozilla Firefox using Weighted GUI and speech data

Comparing the gray boxplots (“speech” and “speech weighted”) in Figure 8-8 shows that, for the *All* baseline applying term weighting improves the median AP and RR and reduces the variation in the plots. For the *Unique* baseline the median AP and RR values remain the same while again removing some variation in the result. Applying term weighting reduces noise in both baselines and improves the AP and RR values for the *All* baseline.

Analyzing the red boxplots (“gui” and “gui weighted”) in Figure 8-8 shows that, for the *All* baseline the median AP and RR values have increased to 100% and the variations are removed by applying term weighting. At the same time, for the *Unique* baseline, although the variations are again removed by applying term weighting, the median values are not improved and had an 18% decrease in both AP and RR values.

A more detailed analysis of these somewhat unexpected decreases in both AP and RR values in the *Unique* baseline shows that this decrease is due to the weighted documents being shorter after the removal of less important words (noise) with a low corpus-wide frequency, resulting in a lower TF_j (the occurrences of words). Also, the *Unique* baseline contains fewer files and therefore fewer TP that can be ranked compared to the *All* baseline.

Our analysis also shows that for the *All* baseline data set, files such as post.php that are commonly ranked at the top of the result set, contain methods that are directly related to features (i.e., GUI items), and therefore have a higher similar term frequency with the screencast artifacts (GUI and speech text).

Results for Mozilla Firefox:

Results for “Default Search Engine” Scenario: After rebalancing the “gui speech” data (blue box plots in Figure 8-10), both the median AP and RR increase. The median AP increases from 0.2 to 0.67 and the median RR increases from 0.22 to 0.5 indicating that the first true positive being ranked now 2 in the top hits of JavaScript evaluations. In contrast, for the C/C++ directories, rebalancing did not improve our ranking results, which is reflected by lower median AP and RR values.

Rebalancing the “speech” data (gray boxplots in Figure 8-10) shows that, for the JavaScript results the median AP and RR values improved by 13% and 67% respectively. While for the C/C++ results, rebalancing the speech data results decreased the median AP and RR values by 5%.

Applying term weighting on the “gui” data (red boxplots in Figure 8-10) for the C/C++ directories, slightly changes the median AP and RR values and the variations (decrease in AP and increase in RR). For the JavaScript directories, the median AP and RR values decreased by 4% and 17% respectively and the variation in the boxplots for the RR values was also reduced.

Based on the median values for all scenarios, term frequency rebalancing improves both AP and RR values for most of the JavaScript results, except for the median AP and RR values of "gui weighted" and the RR value of "speech weighted". For the C/C++ directories, term weighting did not show any improvements for the AP and RR values. Also, similar to the WordPress "*Unique*" baseline evaluations, C/C++ documents contain a low number of words that can be linked to screencast documents' words. Therefore, removing noise from screencasts by rebalancing results in shorter documents, which limits the applicability of LDA and leads to a lower median RR and AP values for the C/C++ evaluations. This contrasts with the frontend development in JavaScript, which handles GUI events and therefore shares more high-level words (i.e., GUI related words) with the GUI of the Firefox browser and the adjustment of term frequencies improved the overall results.

Results for "Import Bookmarks" Scenario: Our evaluations of the blue box plots ("gui speech" and "gui speech weighted") for C/C++ file directories (Figure 8-11) shows that after adjusting the term frequencies, the median AP slightly decreases, while the median RR increases by 25% (from rank 5 to rank 4). Rebalancing of the data reduces the variance in the RR results and the improvement from the rebalancing was significantly larger for the JavaScript compared to the C/C++ results. Also, the variances in both AP and RR results have significantly be reduced and the first true positive's rank improves from rank 5 to rank 2. The median AP of 0.59 and median RR of 0.5 means that while the first true positive is ranked at rank 2 of the results, every 3rd item on the result set is a false positive.

Analyzing the "speech" and "speech weighted" boxplots in Figure 8-11 shows that, for C/C++ directories, adjusting term frequencies reduces the variance in both the AP and RR values and reduces the median values for both boxplots on average by ~1%. For the JavaScript results, term weighting reduces AP and RR variations in the boxplots and increases the median AP value by 14% and reduces the median RR value by 50%. However, the first true positive is still ranked first or second in 50% of the time.

Comparing the "gui" and "gui weighted" (red boxplots) results in Figure 8-11 shows that, for the C/C++ directories, applying term weighting results in a small decrease (2%) in the median AP value with no change in the median RR value. At the same time, the variation in the boxplots is reduced only for the RR values and marginal changes are observed for AP values. The effect of applying term frequency rebalancing for the JavaScript results shows that the median AP and RR values improved on average by 33% and variations in the boxplots are reduced (AP) or removed (RR).

Conclusion**:** Our WordPress analysis shows that, for scenarios with less noisy data (e.g., menu.php), applying a weighting method can improve the AP and RR results for locating project features in the source code, while for scenarios with noisy data, applying term weighting often lowers AP and RR values. Similarly, in Mozilla Firefox, term frequency adjustment improves results when the source code documents contain more high-level concepts (e.g., containing direct references to GUI elements).

RQ10. Are both speech and GUI text data required for feature location?

In what follows, we compare the impact of using either GUI (red boxplots) or speech (gray) screencast data or a combination of both (blue) on the performance of our feature location approach (Figures Figure 8-7,Figure 8-8Figure 8-11,**Error! Reference source not found.**).

Results for WordPress:

Results for "menu" Scenario:

Using speech data only: The gray boxplots (Figure 8-7) show the feature location results our approach achieved for the WordPress *All* and *Unique* baselines. For the *All* baseline, using speech data only, we obtained a median AP of 98% and a median RR 100%. For the *Unique* baseline, both median AP and RR are 50%. Rebalancing the term frequencies shows an improvement in the variations of the AP and RR values for both baselines and improvements for the median AP and RR of the *Unique* baseline. These improvements through the rebalancing are due to the fact that the unbalanced speech data contains a significant amount of noise, which introduces both many false positives and false negatives.

Using GUI data only: For the *All* baseline the first true positive is ranked at the top (median RR = 100%) and in case of the *Unique* baseline the first true positive is ranked at the second top result. The median AP value for the *All* baseline is 0.99 and 0.38 for the *Unique* baseline. Here also, rebalancing the GUI-only screencast shows improvement in the AP and RR values of the *Unique* baseline.

Results for "post" Scenario:

Using speech data only: As Figure 8-8 (the gray boxplots) show, using only speech data without rebalancing for the *All* baseline, the median AP and RR values are 58% and 100% which are as performant as other data types. However, for the Unique baseline these values are lower compared to the other data types. Rebalancing the speech data in the *Unique* baseline results in median of 0 for both AP and RR, which is again due to having shorter documents after rebalancing and having fewer common terms with the source code files in the *Unique* baseline.

Using the GUI data only: The median AP and RR for the *All* baseline is 73% and 100%, which can be interpreted as our approach being able to rank the first true positive at the top of the results and the number of true positives being larger than the number of false positives in the top 10 hits, while the observed performance for *Unique* baseline is lower. Rebalancing of the GUI data reduces the performance for both baselines and both measures while removing the variations.

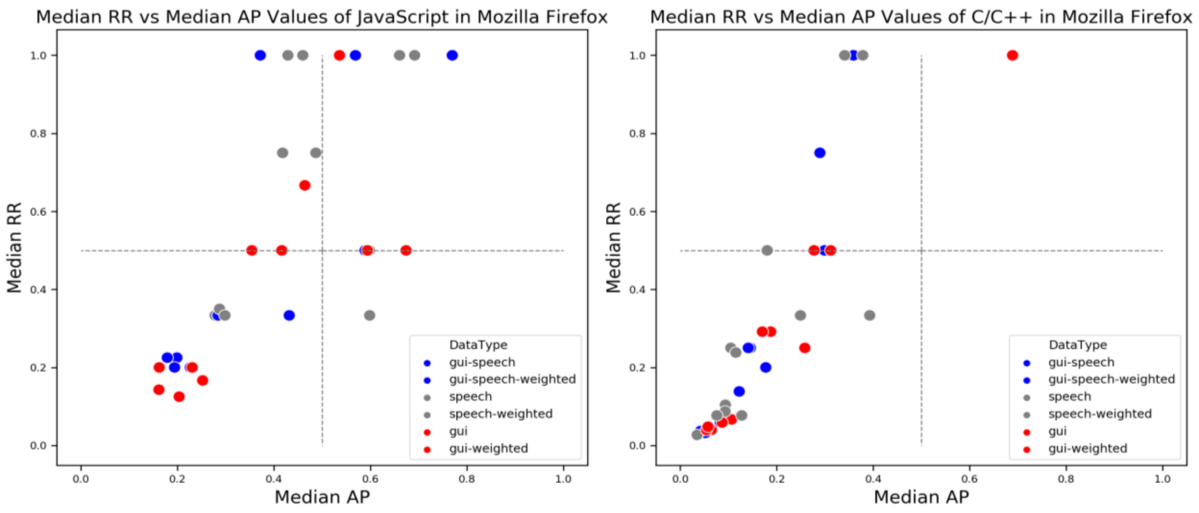


Figure 8-13. Scatter plot of the median AP and RR values for each data type and each language in Mozilla Firefox.

Results for Mozilla Firefox:

Using speech data only: For linking the screencast speech data to C/C++ source code (Figures Figure 8-11 and **Error! Reference source not found.**), adjusting term frequencies in speech data always results in lower median AP and RR values compared to the speech-only data, which is not rebalanced. Removing noise from speech-only documents leads to a poorer performance (yet smaller variance in the results) of our approach since we are now dealing with shorter documents that share less words with C/C++ files, affecting the performance of LDA.

Using GUI data only: The "*Import Bookmarks*", "*Clear History*", "*Set Homepage*", and "*Remove Passwords*" scenarios have the largest number of image frames (Table 8-1) and therefore are expected to have more GUI data. However, Table 8-5 shows that speech data in general outperforms the GUI data. This is due to the fact that the speech data contains more words that are related to the scenarios compared to the noisier GUI data, which contains also many unrelated terms. Also, in most cases, using speech data or combining it with the GUI data and applying term frequency rebalancing, will lead to improve the AP.

For the JavaScript results specifically, Table 8-5 and Figure 8-13 show that using speech-only, rebalanced speech (sw), or rebalanced GUI with speech data (gsw) will result, in most cases, in the highest median RR and/or AP values, except for the "*Import Bookmarks*" and "*Remove Password*" scenarios, where the speech-only data produced the highest median RR value.

Table 8-5. Data sets with the highest median AP and RR values for each project and each baseline (g: gui, s: speech, w: weighted).

Project	**Scenario**	**Baseline**	**Highest Median AP**	**Highest Median RR**
WordPress	Menu	All files	0.88 (gsw, g, gw)	1 (gs, gsw, sw, g, gw)
		Unique files	0.76 (gsw, g, gw)	1 (gs, gsw, sw, g, gw)
	Post	All files	1 (gsw, gw)	1 (gs, gsw, gw)
		Unique files	0.62 (gs)	1 (gs)
Mozilla Firefox	Import Bookmarks	C/C++ dirs	0.23 (s)	0.33 (s)
		JavaScript dirs	0.6 (sw)	1 (s)
	Clear History	C/C++ dirs	0.39 (sw)	1 (s)
		JavaScript dirs	0.66 (sw)	1 (gsw, sw)
	Set Homepage	C/C++ dirs	0.18 (s)	0.5 (s)
		JavaScript dirs	0.77 (gsw)	1 (gsw, gw)
	Remove Passwords	C/C++ dirs	0.69 (gw)	1 (gs, s, gw)
		JavaScript dirs	0.69 (sw)	1 (s, sw)
	Save-to-pdf	C/C++ dirs	0.08 (s)	0.08 (s)
		JavaScript dirs	0.57 (gsw)	1 (gs, gsw, sw)
	Default Search Engine	C/C++ dirs	0.12 (gs, s)	0.24 (s)
		JavaScript dirs	0.67 (gsw, gw)	0.5 (gsw, gw)

The "*Save-to-pdf*" scenario has the lowest observed median AP and RR values for C/C++ files, followed by "*Default Search Engine*" and "*Set Homepage*" scenarios, which have also quite low median AP and RR values for their C/C++ evaluations. A common characteristic of these scenarios is that they contain a significant amount of noise in the GUI data, since the narrator opens as part of these scenarios a web page that contains text unrelated to these scenarios. In all cases with noisy GUI data, the speech data becomes the more reliable data source.

Rebalanced GUI data sets can be beneficial in locating files that contain high-level GUI terms (e.g., *All* baseline in WordPress and JavaScript files in Firefox), but it does not provide any improvement when our approach has to deal with more specific implementation files, such as *Unique* baseline in WordPress or C/C++ in Firefox. In WordPress and Firefox, a combination of GUI data with speech data will in general provide a good performance, while rebalanced speech data in Firefox is overall the most reliable information source.

Conclusion: In WordPress, our analysis has shown that for videos with relatively low noise levels, GUI data by itself can be sufficient for feature location, while speech data will require additional rebalancing. For screencasts with noisier data (e.g., "*post*" scenario), neither speech nor GUI data on their own do perform consistently well. This is the result of having noise (i.e., words that are relevant to the blog post and not to the scenario of creating a blog post) in both speech and GUI data of the scenario. In Mozilla Firefox, speech-only data or rebalanced speech data is the most important information source in locating relevant source code artifacts.

RQ11. What is the performance of the approach in locating source code related to frontend and backend of a project?

As mentioned earlier, our screencast data sets contain feature descriptions typically related to high-level GUI elements. For this research question, we evaluate the applicability of our approach in its ability to link features shown in screencasts to their corresponding frontend and backend implementation. More specifically, we analyze the ability of our approach to trace features shown in screencasts to their corresponding low-level (i.e., backend) and high-level (i.e., frontend) implementation. Next, we report our analysis results for Mozilla Firefox, in which frontend and backend development of features are not only implemented in different architectural layers but also rely on different programming languages (i.e., JavaScript and C/C++).

Mozilla Firefox uses C/C++ for its backend development, which implements the scenario logic and typically has only very few references to any GUI elements of the Firefox browser. As our analysis shows, the median AP values for mapping screencast content to the C/C++ implementation is rather low, since fewer of these high-level terms or GUI-related elements can be directly found in the backend implementation. For the backend implementation, a combination of GUI and speech data produced always the highest median AP or RR values.

JavaScript is used by Firefox to implement and handle GUI events as part of the frontend development and therefore more GUI-related words appear directly in the JavaScript source code. Our study shows that the median AP and RR for the JavaScript data set are always significantly higher than the ones we obtained for the C/C++ data set.

C***onclusion:*** While our approach can provide some traceability between application features shown in screencasts to low-level backend development source code, its accuracy is noticeably higher when applied to frontend implementations that directly implements the high-level or GUI-related aspects of an application.

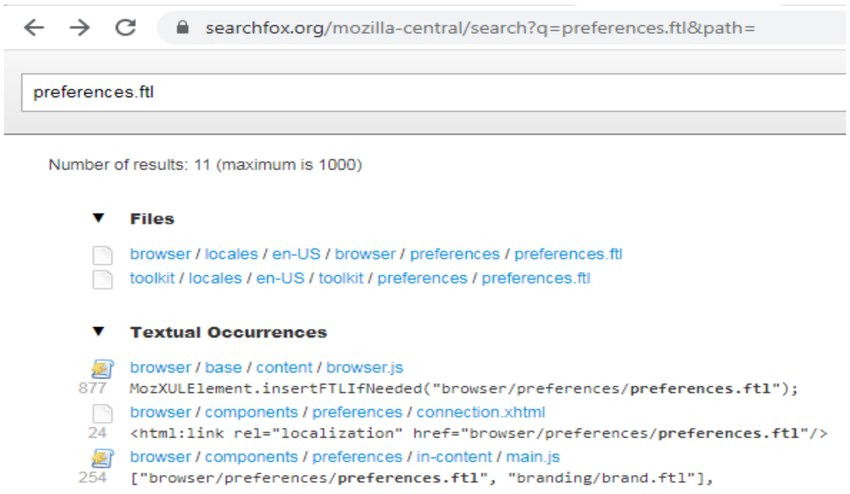
RQ12. How accurately can the approach locate source code directories compared to a guided search approach?

To validate the effectiveness of our approach in locating source code directories that are relevant to a tutorial screencast, we compared our Mozilla Firefox evaluation results not only to a random guess approach (Table 8-4), but also to a basic **guided search** that is more typically used by developers. In such a guided search [99] approach, a developer uses their domain (programming) knowledge and expertise to search for objects. Using this approach, developers narrow their search space more swiftly while locating the source code implementing a feature. For the evaluation of the guided search approach, we again use AP and RR as performance measures.

To simulate this process, we selected the longest screencast for each scenario of our Firefox data set. It should be pointed out that captions of GUI components that are demonstrated in a screencast are not necessarily mentioned or similar to the titles or descriptions of a screencast that are usually used by video portals to search for relevant content. Also, our goal is to simulate a developer who views a screencast and then decides based on the application features demonstrated in the screencast to locate the source code for the viewed features. We therefore manually extracted the text of buttons, labels, and menu items on the GUI that are used (e.g., clicked on) or discussed by the narrator in the screencast while performing an action. Instead of using an IDE or text editor to index the Firefox source code, we used Searchfox[59], a source code indexing tool that is available

[59] https://searchfox.org/

for the code base of Mozilla Firefox, to search for the exact words and patterns that we extracted from the screencasts' GUI widget labels. Searchfox allows for advanced filtering options (e.g., case sensitive matching, regular expression matching, and path filtering) that also help with locating exact words and patterns. In what follows we describe how we simulated a developer's actions who tries to locate the source code for the viewed features on a screencast:



searchfox.org/mozilla-central/search?q=preferences.ftl&path=

preferences.ftl

Number of results: 11 (maximum is 1000)

Files

browser / locales / en-US / browser / preferences / preferences.ftl
toolkit / locales / en-US / toolkit / preferences / preferences.ftl

Textual Occurrences

browser / base / content / browser.js
```
877 MozXULElement.insertFTLIfNeeded("browser/preferences/preferences.ftl");
```
browser / components / preferences / connection.xhtml
```
24 <html:link rel="localization" href="browser/preferences/preferences.ftl"/>
```
browser / components / preferences / in-content / main.js
```
254 ["browser/preferences/preferences.ftl", "branding/brand.ftl"],
```

Figure 8-14. Example results of searching a localization file (preferences.ftl) in Searchfox.

Since we are interested in the frontend and backend feature implementations of Firefox that are developed in JavaScript (i.e., .js) and C/C++ (i.e., .h, .cc, .c, .cpp) source code directories respectively, we limited our search to locate those file extensions.

First, we consider files that contain the exact query string that can result in any file extension (e.g., .xul, .dtd, .ftl, .properties, .js, .xml, .cpp, etc.) as long as the content of the file matches our search criteria. Among the different strings that we use for our query matching are: comments in the code, string literals in localization files, or identifiers. From this initial result set, we store the .js, .cpp, .h, and .c file paths if there is any, and we do not navigate through the located source code files through imported libraries or dependencies, instead we only use the name of the returned localization files[60] (i.e., .dtd, .ftl, and .properties file formats that are used to translate the user interface to different languages, see Figure 8-14) or .xul files to search for other JavaScript or C/C++ paths, since these files contain references to GUI elements or text that is shown in the screencasts and their name is included in our target source code (Figure 8-14).

[60] https://mozilla-l10n.github.io/localizer-documentation/

Next, we further reduce the search space by only selecting source code files that contain the name of the located localization or .xul files and store the directories to these files. We stopped our guided search when a.) no more localization or .xul files could be found or b.) no additional JavaScript or C/C++ source code files were added to our search results. We then calculated the AP and RR values and compared them with the baseline results from the four RQs introduced earlier. Table 8-6 shows the AP and RR values of guided search for JavaScript and C/C++ code at the directory level. We also compared the highest median AP and RR values of the guided search against our feature location approach (Figure 8-15).

Table 8-6. Average Precision and Reciprocal Rank of guided search approach in Mozilla Firefox.

Scenario	Baseline	Guided Search	
		AP	RR
Import Bookmarks	C/C++	0.5	0.5
	JavaScript	0.25	0.2
Clear History	C/C++	0.58	0.5
	JavaScript	1	1
Set Homepage	C/C++	0.14	0.16
	JavaScript	0.86	1
Remove Passwords	C/C++	0	0
	JavaScript	0.28	0.16
Save-to-pdf	C/C++	0.5	0.5
	JavaScript	0.46	1
Default Search Engine	C/C++	0.47	0.33
	JavaScript	0.72	1

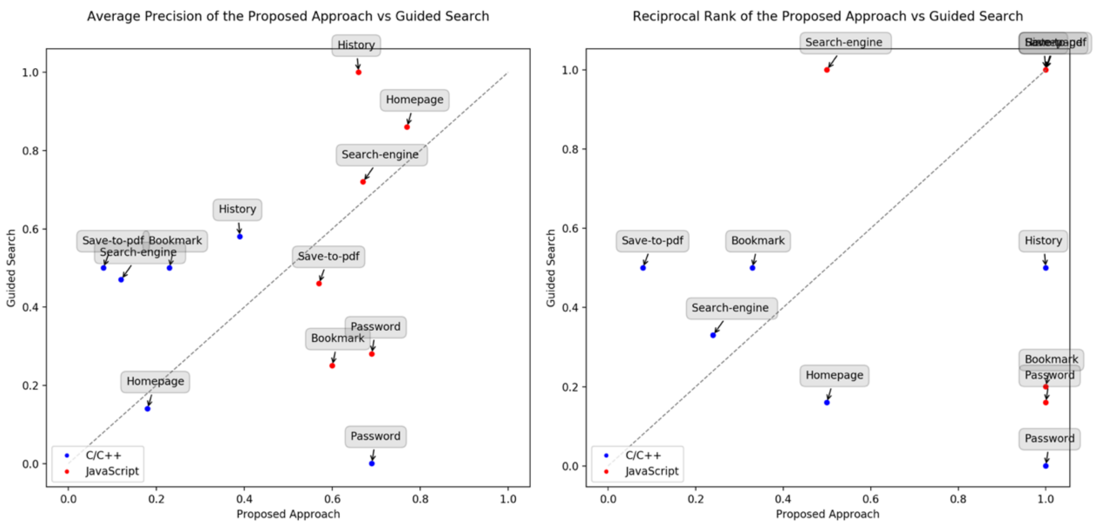


Figure 8-15. Scatter plots of the best median AP and RR values of the proposed approach vs. the AP and RR values of guided search.

Findings: The guided search and our feature location approach **both performed better for the frontend (developed in JavaScript) compared to the backend (developed in C/C++) results**. The scatter plot of the AP values shows that for the "*Default Search-engine*", "*Set Homepage*", and "*Clear History*" scenarios, the guided search outperformed our proposed approach. Two of these scenarios ("*Default Search-engine*" and "*Set Homepage*") contain a significant amount of noise due to having text that appears on web pages that is not relevant to these scenarios. While the guided approach performs better than our approach, an AP above 0.5 for our approach still indicates that the number of true positives is higher than the number of false positives in the top *X*/100 results returned for these three scenarios.

The AP values for C/C++ using our approach are mostly lower compared to the guided search results, except for the "*Remove Passwords*" and "*Set Homepage*" scenario where our approach outperforms the guided search. However, for 8 out of 12 RR values in the case of JavaScript and C/C++, our approach performed equal or better than the guided search. For the AP results, our approach outperformed guided search in 5 out of 12 cases. Furthermore, for all JavaScript directories, the best median AP value for our approach is above 0.5, which means that the number of true positives is larger than the number of false positives in the top 100 results.

Given that our approach attempts to locate only a seed or starting point for searching relevant source code artifacts, the median RR values (which correspond to the rank of the first true positive in the top hits), clearly indicate that our approach significantly outperforms guided search. Furthermore, using our automated approach, such relevant source code directories can be identified/extracted in advance and without the need to watch the screencast or manually interpret feature captions.

Conclusion: Although we only searched for JavaScript and C/C++ files/directories in our guided search and did not include any other call dependencies (files) that might be referenced within these source code files, we still found that the guided search is a time-consuming, manual process that needs to be repeated for each scenario. Even if our approach did not always outperform the guided search, especially for AP, our approach is automated and can therefore significantly speed up the location of relevant source code directories with an accuracy higher or similar to a guided search performed by a developer.

8.4 Discussion

In a case study using 10 WordPress screencasts covering two different feature implementations, we evaluated the applicability of our approach in retrieving relevant results at the top 10 search results. We also performed another case study on 89 Mozilla Firefox screencasts, which cover 6 different Firefox features and in this case the ranking of relevant directories at the top x/100 hits. As part of our evaluation we also studied how modifying term frequencies, based on the importance of the terms, affects the feature location results.

Our case study results show that our approach can semantically link video content (application features) to source code implementing these features. In general, rebalancing of the data improves the quality of results. Also, our approach works best for features that are related to the frontend/GUI-related development of a project. We observed that, speech and image frame data are sufficient to perform our feature location approach and more specifically speech is a very important component of a video and having high-quality speech is an essential information source.

8.5 Threats to Validity

This work is the first attempt to use crowd-based tutorial screencasts for feature location. Our case study results show that the approach can successfully locate relevant source code files in WordPress and source code directories in Firefox using a combination of the speech and GUI, or either data set. Nonetheless, there are some threats to external, internal and construct validity regarding the data sets and the methodology that need to be addressed in future work.

External Validity. In this work, 5 data sets of WordPress tutorial screencasts and 6 data sets of Mozilla Firefox tutorial screencasts, are used. This limited number of videos is due to our selection criteria, such as being uploaded in the same year (likely sharing the same WordPress or Firefox version), being of high quality, and containing both speech and GUI. Also, the curation of the data sets was time consuming, since the results from the text transcription and mined text from the image frames had to be manually verified. For the Firefox data sets, we faced additional challenges in finding tutorial videos with an English speaker narrator. We therefore also included screencasts without any speech. One approach to mitigate this threat would be applying the guidelines for creating screencasts (Section 8.4). Following these guidelines would significantly alleviate these challenges by providing a larger number of higher quality screencasts that are easier to process automatically.

In this work, we mined only textual screencast information from speech and GUI elements. We also captured user actions through textual difference between every two subsequent image frames, and from the source code we only used textual information in the form of comments, identifiers, string literals and file names. Although these information sources already provide promising results, incorporating other information sources, such as the sequence of user events or for example call graphs from the source code should be considered for further improving the approach.

Our exploratory studies consider file and package/directory as granularity level for the WordPress and Mozilla Firefox evaluations respectively. In addition, our feature location approach could be refined to include also method- or even statement-level granularity in the analysis, to provide a more fine-grained analysis.

Construct Validity. Our approach can successfully retrieve the first true positive in the top 10 hits for WordPress, and top *X*/100 hits for Mozilla Firefox. However, the speech-to-text and OCR tools used in this work may have noise or false positives in their output. In our WordPress case study, we partially mitigate this problem by keeping only the words that have a confidence score or accuracy of 0.7 or more in speech documents. While using term weighting can alleviate the effect of such noise in our data sets, the analysis of the text from the transcription or code analysis could be further improved by more advanced speech-to-text or OCR approaches and by using language models and specific language parsers.

One other threat to construct validity is the use of LDA on short documents. Applying LDA approaches that are specifically designed for such documents (e.g., [100]–[102]) could improve the topic models.

Internal Validity. In the case of WordPress, we used two types of baselines to evaluate the performance of the approach. Since the creation of the *Unique* baseline requires us to compare files shared across scenarios, considering more scenarios can potentially reduce the size of unique data sets. In our evaluation, the exact size of the *Unique* baseline does not matter since we compare the performance across the *All* and *Unique* baselines instead of focusing only on the absolute performance values.

In case of Mozilla Firefox, the Gecko profiler does not provide the exact path to the source code files whose methods are executed. In addition, multiple runs of the same scenario are required to have the profiler capture a more complete execution trace for a particular scenario. Using other profiling approaches or tools (e.g., aspect-oriented programming, DTrace[61]) to capture the exact file paths should be considered as future work. Also, we used log likelihood values for different numbers of topics to determine the best number of topics to be used in our experiments. Future work should explore the sensitivity of our approach to the number of topics in more detail.

8.6 Chapter Summary

This chapter presents an application of our proposed methodology for a feature location approach that uses crowd-based screencasts for a given use-case scenario to locate the scenario's source code implementation. We presented our methodology that takes as input textual information (e.g., comments, identifiers, and string literals) from the source code and audio and visual components of screencasts (i.e., speech and GUI text). We apply LDA to structure this information and rank the output seeds in the source code based on the similarity with the feature shown in the screencast. These ranked outputs provide a good starting point for further exploring the implementation of the screencasts' scenarios.

In the following chapter we present a Semantic Web-based application of our proposed methodology that established traceability links between screencasts that demonstrate vulnerability exploitation and their corresponding NVD entry.

[61] http://dtrace.org/blogs/about/

9 A Semantic Web Based Approach for Integrating Screencasts with Security Advisories

9.1 Introduction

Information Security (IS) has emerged as an essential part of software engineering best practices [103]. Specialized advisories or **S**ecurity **V**ulnerability **D**ata**B**ases (SVDBs), such as the National Vulnerability Database (NVD)[62], have been introduced in response to the increasing number of known software vulnerabilities. Vulnerabilities are no longer limited to individual projects or computers, but often affect millions of computers and even complete software ecosystems. SVDBs serve in this context as central repositories for tracking software vulnerabilities and potential solutions to resolve them. However, vulnerabilities and their exploits are only described in these repositories in a textual format, lacking hands-on instructions on how to replicate or fix a known vulnerability.

While these global repositories (e.g., video portals, security vulnerability databases) have been widely adopted by industry to support collaborative software development and knowledge sharing, they also introduce new challenges. These repositories often remain information silos – a situation where a repository is typically not directly linked with another one [104]. For example, source code stored in versioning repositories may contain vulnerabilities already reported in SVDBs. However, without having a link between the source code and vulnerability databases, developers must manually search individual repositories for relevant information or artifacts. A major challenge when establishing traceability links among these repositories is the lack of a common, standardized semantics and knowledge representation that is applicable across repository boundaries.

The research in this chapter illustrates another application of our methodology using different datasets and how results from our approach can be integrated directly within knowledge from other software artifacts as part of a unified ontological representation. This work is a continuation of

[62] https://nvd.nist.gov/

previous work on semantic knowledge modeling using ontologies [105]. In this work, we extend an existing knowledge base with a video ontology **VID**eo **ONT**ology (VIDONT) that allows us to integrating facts extracted from the audio, video (textual cues in image frames) and metadata of screencasts with other software artifacts (e.g., Maven and NVD) in a unified, ontological knowledge representation. More specifically, we introduce bi-directional traceability links from screencasts to NVD security vulnerabilities and show how indirect traceability links between screencasts and Maven project dependencies can be inferred using Semantic Web reasoning. We argue that these links allow us to enrich existing vulnerability information to provide practitioners with different types of vulnerability analysis services. We also discuss several use-case scenarios, by illustrating how developers can benefit from this unified knowledge representation.

It should be noted that the results that we are presenting are currently not generalizable for all types of videos and vulnerabilities, given the diversity of screencasts in terms of their content, length, languages/dialects being used image quality of the videos. Instead, this chapter presents an application of our screencast mining approach and provides results from a case study that we conducted as a proof of concept on a set of screencasts related to software vulnerabilities. We illustrate that it is indeed possible to link screencasts that mention a vulnerability identifier in at least one of its information resources (meta data, image frames, speech) to a vulnerability in a SVDB. We also show that once such direct references to vulnerability identifiers are removed, while more difficult and with lower precision, it is still possible to link vulnerabilities with screencasts.

The main contributions of this work are as follows:

- We introduce our VIDONT ontology that captures the semantics of crowd-based online video repositories (e.g., YouTube).
- We establish bi-directional traceability links between knowledge within the SEVONT and VIDONT ontologies; indirect traceability links are also inferred between VIDONT and SBSON.
- We evaluate the accuracy of these direct traceability links between screencasts and vulnerability information in NVD.

- We perform a case study to illustrate the applicability and flexibility of our modeling approach.

As mentioned before, VIDONT captures the semantics of crowd-based online video repositories and allows us to integrate it with existing SEVONT and SBSON ontologies [105]. This unified knowledge representation will not only provide developers with direct access to vulnerability information described in a screencast content, but also allows for tracing of vulnerability descriptions to relevant screencasts and library dependency information.

Therefore, in this chapter, we aim to answer the following research question:

- **RQ13.** How accurate are the bi-directional links between known vulnerabilities and screencasts published on YouTube, provided by our knowledge model?

In the next section, we provide the background information that is relevant to this work.

9.2 Background

9.2.1 Security Vulnerabilities

In the software security domain, a software vulnerability refers to mistakes or facts related to security problems in software, networks, computers, or servers. Such vulnerabilities represent security risks that can be exploited by hackers to gain access to system information or capabilities [106]. Among these systems, reuse of software libraries poses a significant threat, since vulnerabilities in a single component might affect, through their intended reuse, many different systems across the globe.

Advisory databases (e.g., NVD) were introduced to provide a central place for standardizing the reporting of vulnerabilities and to raise developer awareness about the existence of such vulnerabilities. These advisory databases rely on the Common Vulnerabilities and Exposures (CVE)[63], a publicly available dictionary for vulnerabilities that allows for more consistent and concise use of security terminology in the software domain. Once a new vulnerability is revealed and verified by security experts, information about this vulnerability (e.g., unique identifier, source

[63] https://cve.mitre.org/

URL, vendor URL, affected resources, and related vulnerabilities information) is added to the CVE database. In addition to the CVE entry, each vulnerability will also be classified using the Common Weakness Enumeration (CWE)[64] database. CWE provides a common language to describe and classify software security vulnerabilities based on their type of weakness. NVD, CVE, and CWE can be considered as being part of a global effort to manage the reporting and classification of known software vulnerabilities.

9.2.2 Dependency Management - Maven

Maven, hosted by the Apache Software Foundation, is an open-source build automation tool used primarily for Java projects. In Maven, a software project defines its dependence on any of its artifacts as part of its XML configuration file (also called the POM file), which is stored in the central repository. Upon the build of a project, Maven dynamically downloads the requested versions of all required Java libraries and Maven plug-ins from the Maven central repository into a local cache for use by the project. The Maven Central repository provides open source organizations with an easy, free, and secure way to publish their components for access by millions of developers. The repository is updated with new projects and new versions of existing projects that can depend (in)directly on different versions of the same dependency.

One of the core dependency management features provided by Maven are transitive dependencies. If project-A depends on project-B, which in turn depends on project-C, then project-C is considered a transitive dependent of project-A. Part of Maven's appeal is that it can manage these transitive dependencies and shield developers from having to keep track of all build dependencies required to compile and run an application [107]. As a result, one can now just include a Java library (e.g., Spring Framework) without having to specify the dependencies of that library oneself.

9.2.3 SV-AF: Security Vulnerability Analysis Framework

It is generally accepted that inadvertent programming mistakes can lead to software security vulnerabilities and attacks [106]. Mitigating such vulnerabilities can become a major challenge for

[64] https://cwe.mitre.org/

developers, since not only their own source code might contain exploitable code, but also the code of third-party APIs or external components used by their system. Previous work [105], introduced SV-AF to guide developers in identifying the potential impact of vulnerabilities at both the system and global level.

SV-AF uses a bottom-up modeling approach where system-specific concepts are first extracted, followed by an iterative process of abstracting shared concepts into upper ontologies. To minimize any potential abstraction error, three Ph.D. students from ASEG lab[65] performed a cross-validation of the abstracted ontology layers, reaching an average inter-rater agreement of 95%. The disagreements were resolved through further discussion. The resulting four-layer modeling hierarchy (Figure 9-1) is based on a metadata modeling approach introduced by the Object Management Group (OMG)[66], with each layer providing a different level of abstraction in terms of its purpose and design rationale.

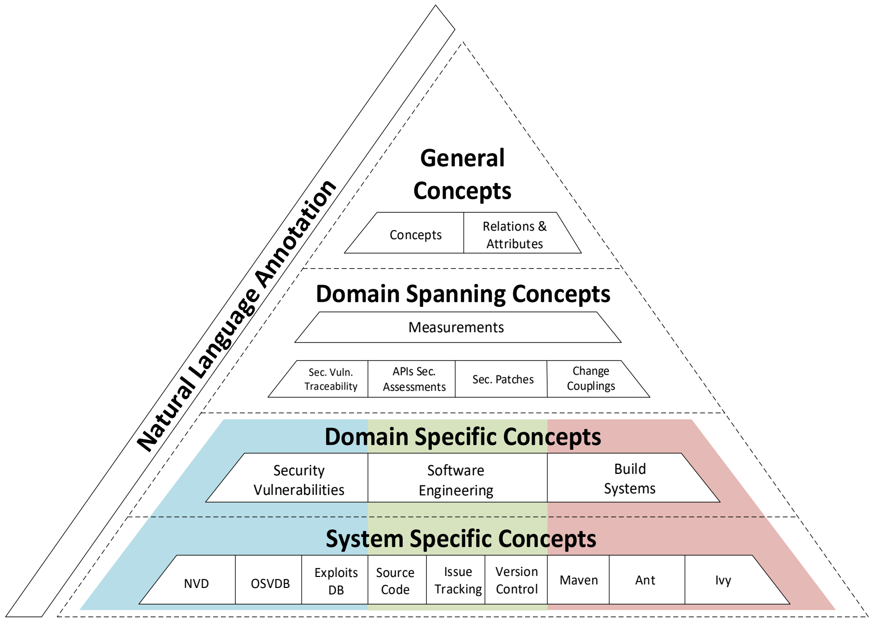


Figure 9-1. The SV-AF Ontologies Abstraction Hierarchies

General Concepts: Classes in the top layer represent omnipresent general concepts found in the software evolution and security domain.

Domain-Spanning Concepts: This layer captures concepts that span across several subdomains (e.g., security databases, video repositories, and source code).

[65] http://aseg.encs.concordia.ca
[66] http://www.omg.org/

Domain-Specific Concepts: Concepts in this layer are common across resources in a domain. At the core of the domain-specific layer, we have several domain ontologies: (1) Software sEcurity Vulnerability ONTologies (SEVONT), (2) Software Evolution ONtologies (SEON) [108] and (3) Software Build System (Dependencies) ONtologies (SBSON).
System-Specific Concepts: Concepts in this layer extend the knowledge from the upper layers to specific vulnerability databases, tool implementations, or programming languages. For example, the Maven system-specific ontology will contain concepts found only in the Maven tool.

SV-AF uses the Probabilistic Soft Logic (PSL) framework [109] to establish weighted links between ontological models of vulnerability databases (SEVONT) and software dependency repositories (SBSON). These traceability links are created based on semantically identical or similar concepts within the different knowledge sources. In this case, similarity among SEVONT-SBSON instance pairs are determined based on the extracted literal information such as name, version and vendor. Using manually defined rules, the PSL framework computes similarity weights between all possible instance pairs in the knowledge base (total of |SEVONT| x |SBSON| instance pairs). These computed similarity weights, based on a given similarity threshold, are used to infer owl:sameAs relations between similar instances found in the two ontologies. The owl:sameAs construct is a built-in OWL predicate used to align two concepts from different ontologies. More details on the ontologies, ontology alignment process, and evaluation of the SEVONT-SBSON alignment can be found elsewhere [105].

9.3 Related Work

Given the diversity in software development processes and their distributed nature, there is a need for knowledge integration and sharing among software artifacts, to improve knowledge reuse and allow for new types of analyses across resource boundaries. Several semantic-web based approaches have been proposed that use ontologies to establish taxonomies in the software engineering domain (e.g.,[110], [111]). These ontologies describe and capture domain knowledge of developers, source code, and other software artifacts. Also, other approaches have been proposed to address the issue of seamless integration of these knowledge resources [108], [112]. While all these approaches aim to promote the inference and integration of new knowledge in an existing knowledge base, they have not considered crowd-based (multimedia) documents as part of their solution space.

Crowd-based documents have become an increasingly popular reference for learning software development/maintenance related skills. This has motivated researchers to embark on research in different areas of extracting information from crowd-based documents and linking these documents to their associated artifacts [113]–[115]. More specifically, recent work has focused on analyzing tutorial screencasts [2], [3], [14], [15], [18], [116]–[119], since screencasts contain tacit knowledge shared by developers and are being frequently produced and used for learning purposes [2].

In what follows, we discuss the work the closest related to ours, on vulnerability dependency analysis, semantic-web enabled software analysis research, as well as mining and linking of crowd-based documents and screencasts.

9.3.1 Vulnerability Analysis in Software Dependencies

Several static vulnerability analysis and detection approaches (tools) exist (e.g. [120]–[124]) that identify vulnerability dependencies in the source code. Common to these approaches is that they identify and track security vulnerabilities and their dependencies at the project level. In contrast, our approach also includes a global dependency analysis of vulnerabilities across project boundaries. Our approach therefore not only allows us to integrate different information resources as part of the analysis, but also provides us with the ability to take advantage of semantic reasoning services to infer implicit facts about the vulnerable code usages within the system, to support bi-directional dependency analysis – including both impacts to external dependencies and vice versa.

Among the existing research most closely related to ours are Cadariu et al. [125], Plate et al. [124], Ponta et al. [126], Decan et al. [127], and Pashchenko et al. [128]. Cadariu et al. [125] [125] introduce in their Vulnerability Alert Service (VAS) an approach that notifies users if a vulnerability is reported for software systems. VAS depends on the OWASP Dependency-Check tool [129]. Plate et al. [124] proposed a technique that supports the impact analysis of vulnerability based on the dynamic analysis of code changes introduced by security fixes. Their work was extended by Ponta et al. [126] to include static analysis of code changes and provide a novel combination of static and dynamic analysis. Among the other related work, Decan et al. [127]

perform an empirical study of the evolution of vulnerabilities within the npm ecosystem and Pashchenko et al. [128] propose an approach for the reliable measurement of vulnerable dependencies in OSS libraries.

Several studies have shown that projects are becoming increasingly susceptible to security vulnerabilities due to the rate at which software libraries are reused within projects. Alqahtani et al. [130] show 750 Maven projects (0.062% of all Maven projects) contain known security vulnerabilities that have been reported in the NVD database. A study by Kula et al. [131] on 4600 GitHub projects showed that 81.5% of them do not update their direct dependencies on vulnerable libraries. A similar study by Eghan et al. [132] on the dependencies of four popular vulnerable projects showed that 36.7% of these projects' dependents updated their dependency to more vulnerable versions. Our approach of providing traceability links between vulnerabilities, project dependencies, and online screencasts addresses the lack of awareness about security vulnerabilities.

9.3.2 Semantic-Web Enabled Software Analysis

Hyland-Wood et al. [133] proposed an OWL ontology of software engineering concepts, including classes, tests, metrics, and requirements. Bertoa et al. [134] focused on software measurement. Witte et al. [61] used text mining and static code analysis to map documentation to source code in RDF for software maintenance purposes. Yu et al. [110] also model static source code information using an ontology and take advantage of SWRL rules to infer common bugs in the source code. Happel et al. [111] proposed KOntoR, which conceptualizes knowledge about software artifacts, such as the programming language used or licensing models. Dietrich et al. [135] developed a tool that scans the abstract syntax tree of Java programs and detects design patterns, described in terms of OWL ontologies, for documentation purposes.

Several researchers (e.g., [112], [136]) have modeled software evolution knowledge found in software repositories as ontologies. Their approaches integrate different artifacts to facilitate common repository mining activities. Tappolet [137] presents a roadmap towards integrating semantics of different software project repositories in three main steps: 1) data representation using RDF/OWL ontologies, 2) intra-project repository integration, and finally 3) inter-project repository integration. Based on these ideas, Kiefer et al. [136] presented EvoOnt, which introduces and integrates the source code, bug and versioning system ontologies. EvoOnt also takes advantage of Semantic Web reasoning services to detect bad code smells, calculate metrics, and to extract data for visualizing changes in code over time. Iqbal et al. [138] presented their Linked Data Driven Software Development (LD2SD) methodology to provide a uniform and centralized RDF-based access to JIRA bug trackers, Subversion, developer blogs, project mailing lists. Wursch et al. [108] presented SEON, a family of ontologies that describe many different facets of a software's life-cycle. SEON is unique in that it comprises multiple abstraction layers. Our core ontologies build upon the SEON knowledge model, which we further extend to support additional software artifacts (e.g., build systems ontology, Video ontology, vulnerability ontologies) and additional reasoning.

Given that these approaches are all based on RDF as a standardized knowledge representation format, we can envision interesting interactions between our knowledge models and the ontologies presented by the other authors. Such extensions could lead to a completely new family of software analysis services or at least simplify the implementation of existing ones.

9.4 Methodology

9.4.1 Overview

The knowledge modelling approach proposed in this chapter establishes traceability links between screencasts (e.g., YouTube) and existing software vulnerability and dependency knowledge found in SV-AF. In what follows, we describe how we extend SV-AF with knowledge from screencasts. Figure 9-2 illustrates our overall research methodology and its major steps.

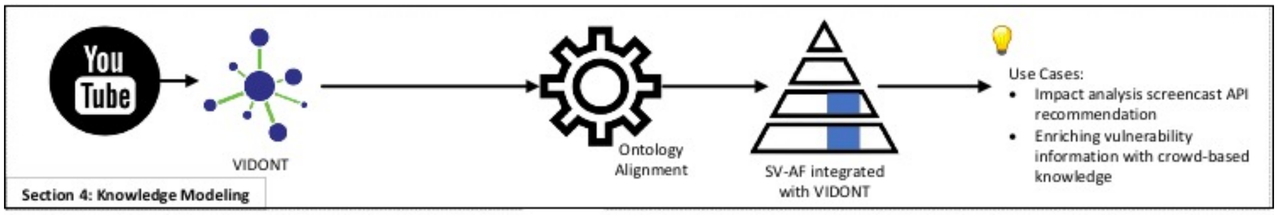
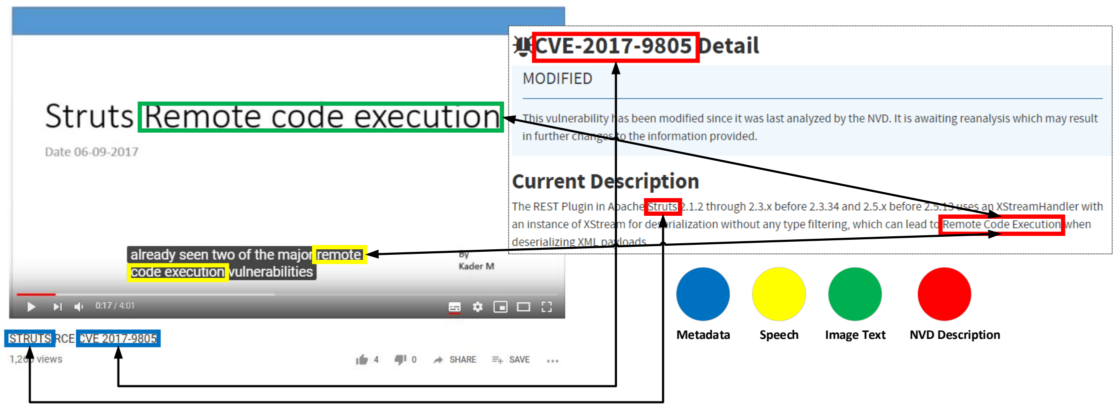

You Tube
VIDONT
Ontology Alignment
SV-AF integrated with VIDONT
Use Cases:
• Impact analysis screencast API recommendation
• Enriching vulnerability information with crowd-based knowledge
Section 4: Knowledge Modeling

Figure 9-2. Overview of the overall methodology. SV-AF is extended with ontologies for the domain of screencast repositories.

9.4.2 Fact Extraction Process

Fact Extraction from Video Artifacts: In what follows, we discuss in more detail the information being extracted and the linking opportunities provided by our approach.

- **Speech:** If a video has a narrator that describes the content of a how-to video, the speech content may share similar words with the image frames shown in the screencast, as well as the relevant entity in NVD.
    - *Information extraction*: Using the approach that is described in Section 6.2.1 we extract the speech content of screencasts.
    - *Linking opportunities*: If the transcription of a video mentions the CVE ID of a vulnerability, one can use this CVE ID to link the video to the relevant NVD entry. However, automatic speech to text tools will not always transcribe such information accurately. To improve the accuracy of traceability links, we also take advantage of Information Retrieval (IR) (see Section 9.3.3) to locate similar artifacts such as the transcribed speech, NVD entries and image frames (GUI text).
- **Image frames:** Figure 9-3 illustrates matching string literals between speech, image frame text, and vulnerability description in NVD.
    - *Information extraction*: Using the approach introduced in Section 6.2.3, we extract the GUI text of the image frames of each screencast.
    - *Linking Opportunities*: If an image frame contains a CVE ID, OCR will be able to extract this CVE ID with high accuracy. Such an extracted CVE ID can then again be linked to the NVD repository. In addition, similar to spoken text, we can use the text on the GUI as an information source to link a video with NVD entries. With each image frame, we also have its position (timestamp) within the video, which allows us to directly link external knowledge resources to the relevant part of the video.



CVE-2017-9805 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Current Description

The REST Plugin in Apache Struts 2.1.2 through 2.3.x before 2.3.34 and 2.5.x before 2.5.13 uses an XStreamHandler with an instance of XStream for deserialization without any type filtering, which can lead to Remote Code Execution when deserializing XML payloads.

Figure 9-3. Similar string literals in speech, image text, and NVD description

- **Metadata:** We also extract and analyze metadata that is included in each screencast.
  - *Information extraction:* Among the metadata to be extracted are video title, description, published date, comments, number of likes and dislikes, number of views, closed captioning (if enabled) and other information related to the screencast [14]. We use youtube-dl[67] to extract this data.
  - *Linking opportunities:* This metadata often contains important information that can be added to the video documents to support the linking of screencast content to NVD vulnerabilities. For example, the publication date can be used to help identify vulnerabilities discussed in a video.

It should be noted that when preprocessing text extracted from the transcribed speech, image frames, and metadata, we do not remove tokens that combine characters and numbers since CVE IDs are composed of these characters (Figure 9-3 contains a sample CVE ID). We, however, apply tokenization, removal of stop words and punctuation, and perform stemming to clean up the data before populating the ontology in Section 9.3.3.

Fact Extraction from Vulnerability Artifacts: Security databases (e.g., NVD) provide a central place for reporting vulnerabilities affecting existing software applications and systems. Extracting facts from NVD consists of downloading and parsing its vulnerability XML feeds.

- *Information extraction*: In NVD, a vulnerability is identified by its unique CVE ID. NVD also captures additional vulnerability details, such as: vulnerability disclosure date, severity score, vulnerability summary, and sources of information that demonstrate the vulnerability.

[67] https://ytdl-org.github.io/youtube-dl/index.html

Each vulnerability has a list of affected products associated, described by its Common Platform Enumeration (CPE)[68] a standard machine-readable format for encoding names of IT products and platforms.

- *Linking opportunities*: Screencasts may contain some of the information in published vulnerabilities such as the CVE ID and vulnerability summary. The CVE ID may be found in the video's title, description, speech, or image content. Also, the same words that are used in the description of a vulnerability in NVD may be used in the screencast data (i.e., speech, images, metadata). Therefore, we extract this information from the vulnerabilities to be further used in our linking approach.

Fact Extraction from Maven Artifacts: Maven artifacts describing the dependencies used by a project are stored in the Maven Central Repository. Extracting Maven facts consists of transforming the Maven central repository index into a list of GAV (groupId, artifactId, and version) coordinates - the three required elements to describe every project. The groupId is a unique name amongst an organization, and the artifactId is generally the name by which the project is known. Several projects developed by the same organization or under the same parent project will have the same groupId. The POM file for each GAV entry is parsed and populated into our knowledge base.

- *Information extraction*: As stated above, each project release in Maven Central is identified by its unique GAV coordinate. The POM file for each GAV entry captures various project details, such as the project name, dependencies on other project releases, organization, developers, release date, and links to its repositories (e.g., issue-trackers and CVS).
- *Linking opportunities*: The details captured within each project's POM file provide several linking opportunities to existing knowledge bases. For example, project identification details such as the GAV, project name, or URL can be used to identify project entities in the textual descriptions of videos based on string matching. They can also be used to identify products affected by vulnerabilities within the NVD dataset (see [139]). Furthermore, project dependency information (using the <*dependency*> tags) can be used to establish links to the text extracted from source code samples in videos.

[68]Common Platform Enumeration – http://cpe.mitre.org

9.4.3 Knowledge Modeling – Extending SV-AF with Knowledge from Screencast Repositories

In this section, we introduce VIDONT, our ontology to capture the semantics of crowd-based online video repositories (e.g., YouTube). As part of our knowledge modeling, we then integrate VIDONT with our existing SV-AF ontologies at the domain level. The integration of VIDONT and SV-AF allows not only for knowledge sharing and reuse across repository boundaries by eliminating traditional information silos these resources have remained, but also allows for novel types of vulnerability analysis and documentation approaches.

Figure 9-4 provides an overview of the main classes and object properties across all layers of our knowledge model that are used for linking screencasts to project vulnerabilities. To improve the readability of this chapter, we denote OWL classes in *italic* and properties are underlined. The core concepts used in our model are *Vulnerabilities*, *Videos*, and *APIs*. A project version that is released to the public or customer is referred to as a *BuildRelease* (a *BuildRelease* can dependOn *APIs* from other *BuildReleases*). Different project metadata are captured using the hasName, hasDescription, hasURL, and hasVersionNumber properties. Whenever such a project is identified to be affectedBy a *Vulnerability*, a description of a vulnerability is publicly disclosed in repositories such as NVD. Each publicly disclosed vulnerability is issued a unique ID, captured by the hasVulnerabilityID property.

Details of existing vulnerabilities, their exploits, and how such exploits can be mitigated are describedBy *Videos* provided by *Publishers* using video repositories such as YouTube. A publisher isA type of project *Stakeholder*. In our model, published videos have information encoded as *Speech* and *Image* frames. Videos have metadata associated such as title, description, and published date. Keywords and tags for a video are captured through the label property. Typical for crowd-based videos is that they allow for *Comments* and discussions from other users and viewers. The link between videos and vulnerabilities can be established using a *SimilarityMeasure* that measures the relevance of a given video in terms of covering a known vulnerability. For a complete description of our ontologies, we refer the reader to our earlier work [105], [139].

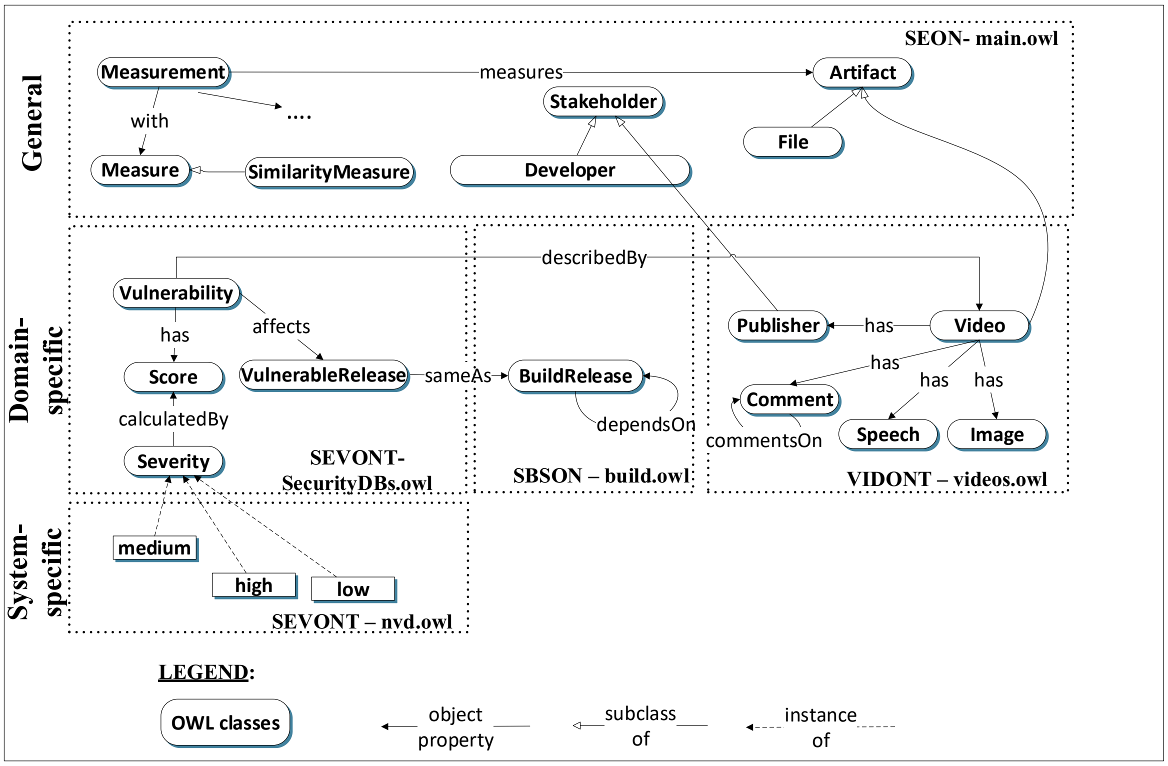


Figure 9-4. An overview of concepts and properties in our integrated knowledge model

Ontology Alignment and Knowledge Inferencing: To further improve the knowledge integration between the VIDONT, SEVONT and SBSON ontologies, we establish semantic traceability links through ontology alignment. During ontology alignment, identical or equivalent concepts, properties or facts in multiple ontologies are identified by analyzing the data captured as part of ontology descriptions (e.g., labels, comments, attributes and types, relations with other entities) and by using logical reasoning to infer correspondences. By establishing these links, we can reduce the semantic gap between the individual ontologies, which is an essential prerequisite for providing a unified knowledge model.

Table 9-1. Ontology namespaces

Namespace	URL
RDF	http://www.w3.org/1999/02/22-rdf-syntax-ns#
OWL	http://www.w3.org/2002/07/owl#
SBSON	http://aseg.cs.concordia.ca/segps/ontologies/domain-spanning/2015/02/build.owl#
SEON	http://se-on.org/ontologies/general/2012/02/main.owl#
SEVONT	http://aseg.cs.concordia.ca/segps/ontologies/domain-spanning/2015/02/vulnerabilities.owl#
VIDONT	http://aseg.cs.concordia.ca/segps/ontologies/domain-spanning/2018/01/video.owl#
MEASURE	http://se-on.org/ontologies/general/2012/02/measurement.owl#

In what follows, we discuss in more detail how we use Semantic Web inference techniques to establish these additional semantic traceability links between our ontologies. The reasoning services allow us to discover additional relations to other knowledge and facts captured in our knowledge base. It should be noted that we omitted the ontology namespace prefixes (summarized in Table 9-1) from our illustrative queries and rules shown in this section to improve their readability.

SEVONT and VIDONT Ontology Alignment. Screencasts and other tutorial videos often contain references and keywords related to a vulnerability, such as the CVE ID in the video title, description, speech, or text encoded in image frames. Our alignment process links instances in the two ontologies based on the presence of such references and keywords. These vulnerability references are used as tags during the video data extraction process. For example, the triple *<https://youtu.be/Wewl5fAhnXA> rdfs:label "CVE-2015-0096"* represents a video resource tagged with a CVE ID that was found in its title. This knowledge can now be used to perform terminology matching, by aligning instances from the vulnerability and video ontologies. For the alignment, we use the Semantic Web Rule Language (SWRL)[69] to create the rule in Listing 1, which can now infer links between vulnerability and video instances.

```
Video(?video), label(?video,?label), Vulnerability(?vuln),
hasVulnerabilityID(?vuln,?label)
→ describedBy(?vuln,?video)
```

Listing 1: SWRL rules for aligning CVE facts with the video ontology.

However, the alignment rule in Listing 1 only applies when a vulnerability's CVE ID is explicitly mentioned in a video. To support cases where no CVE ID is explicitly mentioned, we complement our alignment approach with the BM25 probabilistic relevance model [92]. BM25 is a popular model used in Information Retrieval (IR) to rank a set of documents based on their relevance to words in a given query. It is based mainly on the term and document frequency measures. Given a query, Q, containing keywords $q_1,\ldots, q_n$, the BM25 score of a document, D, that measures the similarity between Q and D is calculated as:

[69] https://www.w3.org/Submission/SWRL/

$$score(D,Q) = \sum_{i=1}^{n}\left(\frac{N}{df(q_i)} * \frac{tf(q_i,D)*(k_1+1)}{tf(q_i,D)+k_1\left(1-b+b*\frac{|D|}{avgdl}\right)}\right), \qquad (9\text{-}1)$$

where $tf(q_i, D)$ is $q_i's$ term frequency in D, $df(q_i)$ is the number of documents containing q_i, |D| is the length of D in words, avgdl is the average length of all documents used in the relevance scoring, N is the total number of documents indexed, and k_1 and b are free parameters used to scale the document term frequency and document length respectively.

BM25 in Ontology Alignment: Using BM25 (Formula 9-1), we can derive the relevance score for each vulnerability-video pair. These derived scores become confidence measures for the alignment links between a vulnerability-video pair, and are later materialized into our knowledge base using the *SimilarityMeasure* class and the measure:measuresThing and measure:hasMeasureValue properties (see Figure 9-5). The measure:measuresThing property identifies the vulnerability and video facts that are being compared, while the measure:hasMeasureValue property stores the numeric similarity value. With this new knowledge, the SWRL rule in Listing 2 can be executed to establish vidont:describedBy relations (similar to those inferred in Listing 1) between vulnerability and video instances captured by the SEVONT and VIDONT ontologies.

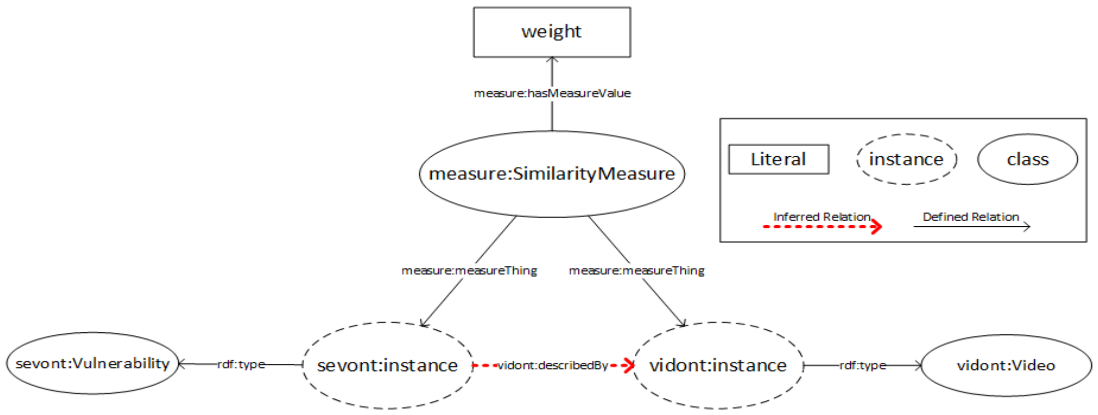


Figure 9-5. Ontology alignment based on BM25 relevance scores

```
Video(?video), Vulnerability(?vuln), measuresThing(?measure, ?video),
measuresThing(?measure, ?vuln), hasMeasuresValue(?measure, ?relevanceScore),
swrlb:greaterThan(?relevanceScore, thresholdValue)
→ describedBy(?vuln,?video)
```

Listing 2: SWRL rules for aligning CVE facts with the video ontology based on their relevance score

Given our populated ontologies, it is now possible to infer implicit knowledge using the describedBy link between a vulnerability and a screencast. These links are instances where either a CVE ID is explicitly mentioned in a video (Figure 9-6a) or the link is inferred when the similarity measure between the vulnerability index and video text queries is within the specified threshold (Figure 9-6b).

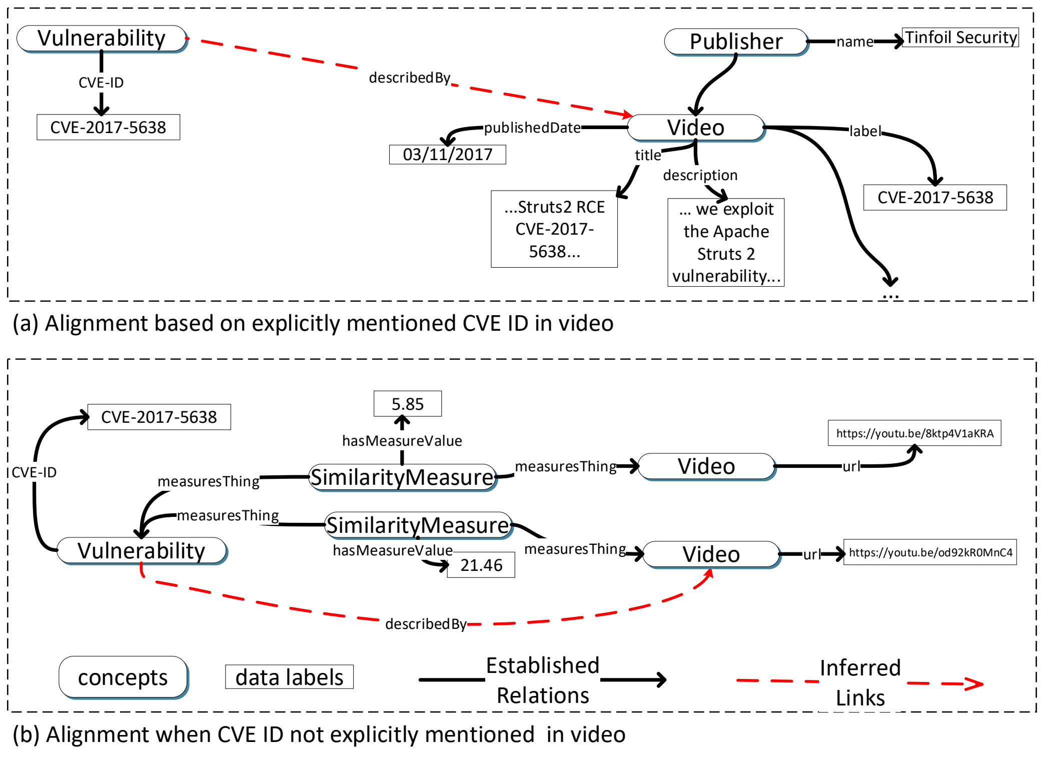


Figure 9-6. Inferring the describedBy link between vulnerability and screencast instances when a CVE ID is (a) explicitly mentioned in a video, and (b) not mentioned

SBSON and VIDONT Ontology Alignment. Having the SEVONT-VIDONT alignment and existing SEVONT-SBSON alignment (see Section 9.2.3), we are now able to infer indirect traceability links between VIDONT and SBSON by taking advantage of Semantic Web inferencing services such as owl:sameAs and *owl:TransitiveProperty*.

Same-As Inference: The same-as inference is commonly used to align two semantically equivalent concepts or individuals. For example, in our prior work [105], [139], we used the owl:sameAs property to align vulnerable project releases from the SEVONT ontology to their corresponding instances in SBSON ontology based on a similarity threshold. Using the SPARQL query shown in Listing 3, we can now take advantage of this knowledge to establish indirect links from videos to project releases in the SBSON ontology (e.g., retrieve metadata of projects affected by a vulnerability described in a video).

Transitive closure inference: A relation R is said to be transitive if R(a,b) and R(b,c) implies R(a,c); this can be expressed in OWL through the *owl*:*TransitiveProperty* construct. In SBSON, we define a dependency between projects using the bi-directional transitive seon:dependsOn property to allow us to retrieve a list of all releases that have a direct or transitive dependency on a specified project release, and vice versa. Using the SPARQL query in Listing 4, we can now establish indirect links from videos to this project dependency knowledge in the SBSON ontology (e.g., to verify if a project transitively uses another project affected by a vulnerability described in a video).

```
SELECT ?video ?release ?name ?desc ?version ?url
WHERE {
   ?vulnerability vidont:describedBy ?video.
   ?release a sbson:BuildRelease.

   #using the same-as links between SEVONT and SBSON release
instances
   ?vulnerability sevont:affectsRelease ?release.
   ?release seon:hasName ?name ; sbson:hasVersionNumber
?version ;
         seon:hasURL ?url ; seon:hasDescription ?desc.
}
```

Listing 3: SPARQL query returning videos and related project details

```
SELECT ?video ?release ?dependent
WHERE {
   # using the transitive inference to detect dependencies on a vulnerable
release with a video description
   ?dependent sbson:hasBuildDependencyOn ?release option (transitive).
   ?vulnerability vidont:describedBy ?video.
   ?release a sbson:BuildRelease.
   ?vulnerability sevont:affectsRelease ?release.
}
```

Listing 4: SPARQL query returning videos, related projects, and their dependencies

9.5 Case Study: CVE-2017-5638

In what follows, we report on results from a case study that we conducted to illustrate the applicability of our approach and to answer our research question from Section 9.1.

9.5.1 Case Study Setup

Dataset: We use a publicly disclosed vulnerability, CVE-2017-5638[70], which has been reported in the NVD repository as a vulnerability affecting several Apache Struts[71] releases. Our dataset for this case study includes data from NVD, Maven Central, and YouTube.

Our vulnerability dataset consists of all NVD vulnerability XML feeds published since 1990. The dataset includes 74,402 unique vulnerabilities that affect 186,212 projects. Our Maven Central dataset consists of 178,763 unique projects with 1,849,756 releases[72]. For our video dataset, we downloaded 48 YouTube videos related to the CVE-2017-5638 vulnerability.

The videos were selected using search queries developers would use when manually searching YouTube for videos related to this specific vulnerability: "CVE-2017-5638", "CVE-2017-5638 Apache Struts", "input validation vulnerability Apache Struts", and "input validation vulnerability exploitation", with input validation corresponding to the CWE vulnerability category. From these search results, we selected 48 videos that also matched additional selection criteria such as video length and video resolution. We only selected screencasts with a length between 20 seconds and 12 minutes to ensure that they contain sufficient data in terms of speech and video content and where the screencast was recorded as High Definition (HD) to allow for a more accurate information extraction from the image processing step.

Of the selected 48 videos, 39 videos explicitly mention the CVE ID in either their title, description or video content. The 9 videos that do not mention explicitly the CVE ID were manually evaluated to ensure they were related to the CVE-2017-5638 vulnerability. Among the videos only 13 videos have an English narration. Table 9-2 provides an overview how many of the videos in our dataset capture the CVE-ID in the different video artifacts (speech part, image frames, video title, and

70 https://nvd.nist.gov/vuln/detail/CVE-2017-5638
71 https://struts.apache.org/
72 Dataset last updated 2017-10-23

video description), with (✓) indicating that the CVE-ID was explicitly mentioned and (×) indicating the CVE-ID was not mentioned at all.

Table 9-2. Classification of the video dataset based on the presence of CVE ID

Presence of CVE ID				# of videos
title	description	image frames	speech	
×	×	×	×	9
✓	×	×	×	5
×	✓	×	×	3
×	×	✓	×	2
✓	✓	×	×	2
✓	×	✓	×	11
×	✓	✓	×	5
✓	✓	✓	×	9
✓	✓	✓	✓	2

Applying our Methodology: For all screencasts, we downloaded the audio (speech part) and automatically transcribed the speech using IBM Watson's Speech-To-Text (STT) service. We used FFmpeg to extract image frames from the downloaded videos at a rate of 1 frame per second (to reduce the number of continuous frames with duplicate content). Next, we manually checked and removed image frames at the beginning and end of the screencasts which contain non-relevant information (e.g., greetings, introducing the YouTube channel or video creator, inviting people to like/subscribe, etc.) to reduce the amount of noise in our screencast data set. It should be noted that this could be also be automated (e.g., by removing the first and last 10 seconds of each video. Then we use Google Vision API's text recognition service to perform Optical Character Recognition (OCR) and automatically extract all text from the remaining image frames.

We also automatically extracted the title, description, publication date, and publisher information from the metadata provided with each video (if applicable) using the youtube-dl tool. Using a regular expression, we then searched for the CVE ID in the speech, image text, and metadata to label each video with a CVE ID and populate our knowledge base.

As part of the next processing step, we create an inverted index using the extracted text from the vulnerability dataset as our document corpus. More specifically, for this index, we treat each vulnerability as an individual document, with its own document id and its textual content being a bag of words. We then apply a simple preprocessing step consisting of case-folding, stop word removal, and stemming before indexing the document. For our case study, we use the extracted text from the metadata, images, and speech of our videos as query terms and populate the *SimilarityMeasure* instances (using the BM25 formula discussed in Section 9.3.3) for the Top 10 ranked vulnerabilities returned by each search query. Applying the semantic rules introduced earlier (Listings 1 and Listing 2), we can now automatically infer bidirectional traceability links from the screencast instances to their related vulnerability instances.

Evaluation Measures: For our case study, we evaluate the linking accuracy of our approach. The objective of this evaluation is to validate whether our modeling approach is indeed capable of inferring hidden and indirect links between videos and other software repositories.

As part of our evaluation, we compared the retrieved ranking results against our video dataset oracle (baseline) that was created using manually selected and verified videos covering the CVE-2017-5638 vulnerability. Since in most cases we only have one relevant result (vulnerability) that matches a query (video), we used the position of the first true positive in the search result ranking as our assessment criteria. For the evaluation, we used the Reciprocal Rank (RR) measure. RR considers the position of the first true positive in the search results [140] and is calculated using the following formula:

$$RR = \frac{1}{rank\ of\ the\ 1st\ TP} \qquad (9\text{-}2)$$

For the second part of the study, we used BM25 [24] to evaluate the ability of our approach to rank documents that do not contain an explicit mentioning of the CVE ID in neither their metadata nor screencast content.

9.5.2 Case Study Results

RQ13. How accurate are the bi-directional links, between known vulnerabilities and screencasts published on YouTube, provided by our knowledge model?

In what follows, we evaluate the linking accuracy of our approach, by comparing it with our baseline (our initial labeled video dataset) of 48 vulnerability-screencast pairs. The objective of this section is to illustrate that our modeling approach is indeed capable of inferring hidden and indirect links between video and other software repositories with sufficient accuracy.

For our evaluation, we use precision and recall measures, with true positives being the number of vulnerability-screencast pairs correctly matched, while false positives correspond to the number of vulnerability-screencast pairs incorrectly matched. For recall, false negatives correspond to the number of correct vulnerability-screencast pairs that were not identified by our approach.

Evaluation of CVE ID Alignment: Figure 9-7 shows the results of our approach when using our 39 videos that explicitly mention a CVE ID in their title, description or content. The results show that, our approach achieves, as expected, high precision values of 1.0 and 0.96 when a video explicitly mentions a CVE ID in its title or description. During our analysis, we also observed that the text extracted from the image frames explicitly mentioned several other CVE IDs (usually related to the CVE-2017-5638 vulnerability) which resulted in a low precision of 0.30, which also affects the precision of our approach when all information sources are used together.

A manual inspection of the inferred links revealed that the lower recall (Figure 9-7) for the individual information resources in a screencast is due to inaccuracies found in the transcription, text of the image frames, or the metadata of the screencasts. Also, depending on the accuracy of the speech to text transcriptions, numbers mentioned within the CVE ID in the speech part of a video are often erroneously transcribed into another word(s), which no longer allow us to link these CVE IDs with their NVD counterparts. However, if all information resources are combined, we were able to achieve a higher recall of 0.81. The evaluation also highlights the impact of different information resources and their trade-off on precision and recall. Using only available title and description information, our approach can achieve high precision, but recall will be quite low. In contrast, by taking advantage of all available information resources, our linking approach will achieve a much higher recall but also a significant lower precision, due to many false positives.

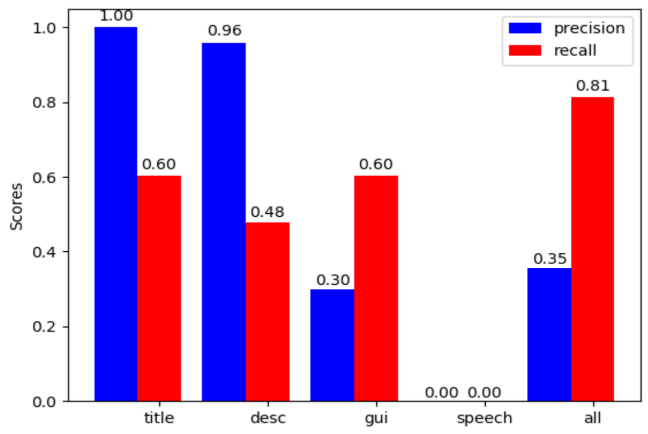


Figure 9-7. Precision and recall of our CVE ID alignment, using only video title, description, image frames (“gui”), speech, or all the above (“all”)

Evaluation of BM25 Relevance Alignment: As our previous evaluation showed, our approach can link in most cases successfully NVD and Video content if a CVE ID is present. In this part of our evaluation, we focus on the ability to rank documents that do not contain any explicit mentioning of the CVE ID in either their metadata or the screencast content with the description found for the vulnerability in the NVD repository. As part of this evaluation process, we are interested in the ranking of relevant (true positive) results in the top hits of the result set. For the evaluation, we first manually searched and removed all instances of CVE IDs found in the 48 videos of our video dataset and then re-applied our linking approach on this data set. For the evaluation, we compared the linking results, with the results from our initial (labeled) video dataset. Figure 9-8 shows that the median RR for our dataset without any explicit mentioning of the CVE ID is 0.01, which means that our top 10 results rarely contain any true positive. The main reason for the poor performance of our approach is that the vulnerability related text in the analyzed videos is too generic and inclusive, to allow for a relevant matching between the video content and a specific CVE ID (and its description).

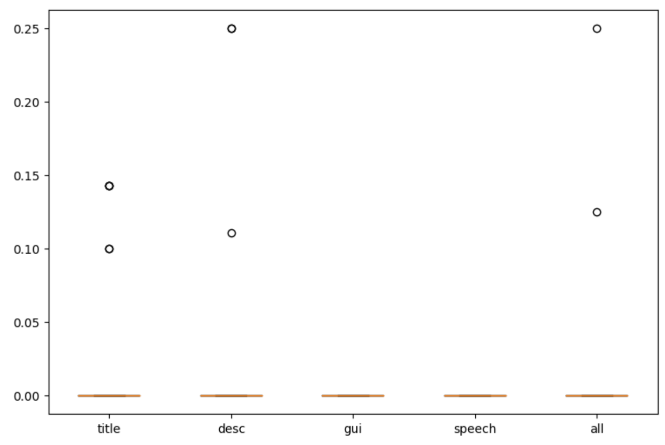
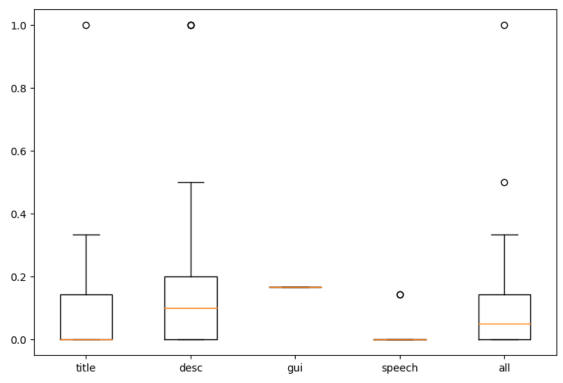


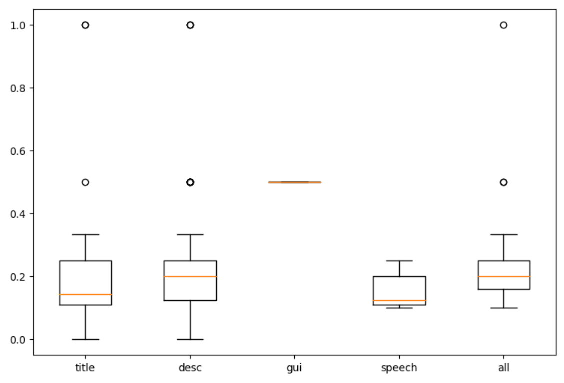
Figure 9-8. Reciprocal Rank results for BM25 alignment evaluation

To further improve the ranking results of our approach, we narrowed the vulnerability search space by taking advantage of available semantic information such as affected projects and vulnerability category keywords found in the video text. We used DBPedia Spotlight[73], a tool which automatically annotates mentions of DBPedia[74] resources in a given text, to identify any reference to a software product (which are represented as entities of DBpedia's Software class) in our video and vulnerability datasets. DBpedia is a linked-data knowledge base that provides a rich source of RDF descriptions of million entities such as companies and products. Using this approach, we can reduce our vulnerability search space (from 74,402 to 73 vulnerabilities), by including only those vulnerabilities which were annotated with the same software products shown in the videos of our dataset. As the RR results in Figure 9-9a show, our semantic alignment process shows a significant improvement for RR values when all our information sources are used on the reduced number of vulnerabilities. Using only the text from the image frames (i.e., GUI text) resulted in the highest RR value of 0.16. This can be interpreted as follows: 100% of the time, the first true positive (matched vulnerability) was ranked in the 6th position in the result set. The average median RR value over all boxplots increased from 0.01 to 0.11, representing a 9th position rank for the first true positive, 50% of the time. A further manual analysis of the results showed that 3 videos are ranked in the 1st position when we used only video descriptions, and 1 video is ranked in the 1st position when we used either the video titles only or all the combined information sources as our query terms during the alignment.

[73] https://www.dbpedia-spotlight.org/
[74] https://wiki.dbpedia.org



(a) RR results after "product" filtering of search space



(b) RR results after "product and CWE category" filtering of search space

Figure 9-9. Reciprocal Rank results after narrowing the vulnerability search space

We further reduced the vulnerability search space (from 73 to 14 vulnerabilities) by performing a hierarchical search, first by comparing video text with vulnerability category (CWE) descriptions and then comparing videos with vulnerabilities within the top 10 ranked category results. Using this hierarchical search approach, we now only include vulnerabilities of the same category. Figure 9-9b shows the RR results after applying this filtering by "product and CWE category". The average median RR value of all the boxplots improved to 0.27, indicating that the 1st true positive is now ranked 50% of the time at the 4th position in the results set. Also, the highest median RR value increased to 0.5, indicating that the 1st true positive is now 100% of the time ranked in the 2nd position text when only text from image frames (i.e., gui text) are used. However, we did not see any significant improvements in the number of true positives ranked in the first position (p-value = 0.81). These initial results look promising and provide avenues for future enhancements to our approach.

A key finding from our assessment is that when a CVE ID is not explicitly mentioned, text from video image frames is the most informative (relevant) resource to be used for linking videos to vulnerabilities.

We also believe that similar to how traceability links are generated between unannotated commits and issues, techniques such as comparing video and vulnerability publication dates and identifying the presence of the vulnerable source code elements (class or method names) within the content of videos can be used to improve the accuracy of our proposed approach. Also, creators of vulnerability related videos should follow more rigorously existing screencast best practices (e.g., [2]) and provide more precise and concise vulnerability information in their videos to allow for easier integration (linking) of video content with other software artifacts. For example, CVE IDs, name of the affected project being demonstrated, and keywords from the vulnerability classification should be explicitly included in the video.

Another finding from our case study is that both short and concise information sources such as video titles and description, and verbose sources such as video speech and text displayed in image frames are important for our linking approach. For example, Figure 9-9a and 9-9b above show that the results obtained by using only text extracted from image frames as queries are ranked higher than the other information sources on average, probably because they contain more content than the short descriptions/titles.

9.5.3 Discussion

As our evaluation shows, although videos and NVD contain different types of information describing known vulnerabilities, similarities and semantics captured by these artifacts can be used to allow for the linking of these knowledge resources. Our case study illustrates, that using a unified Semantic Web modeling approach can indeed allow for the integration of these heterogeneous knowledge resources. In addition, this unified knowledge representation allows us to transform the traditional information silos these artifacts have remained into information hubs, where knowledge can be seamlessly shared and reused across resource borders. Our ontology-based knowledge model also provides a machine-human readable representation that supports incremental knowledge population based on the Open World Assumption.

Furthermore, reusing the previously established bi-directional traceability links from Maven Central to NVD security vulnerabilities [105], [139] allows us to infer indirect traceability links between screencasts and Maven project dependencies. The Maven repository includes many vulnerable projects/components that are commonly reused within existing projects in the Maven ecosystem. More specifically, while a project might not have any direct vulnerability reported in the NVD database, it can still be potentially affected indirectly through its dependency on other (external) vulnerable libraries and components. Providing traceability links between screencasts describing vulnerabilities found in projects (or their direct/indirect dependent components) can provide developers with additional insights in the potential threats their project is exposed to and help them to mitigate the security issues.

To illustrate how our knowledge model can be used to infer new knowledge, we revisit our motivating example (scenario #1) from Chapter 2, where Bob is following the instructions shown in a screencast implementing his project without being aware that the component that is explained in the screencast is using (dependent) a vulnerable third-party API. Given our knowledge model, we are now able to first identify vulnerable APIs or components Bob's project might directly or indirectly depend on and then recommend him a screencast that illustrates these known vulnerabilities. For example, Table 9-3 shows the four most commonly used Apache Struts releases affected by the vulnerability CVE-2017-5638 and the number of projects dependent on them, based on Listing 4 (Section 9.3.3). As shown in the table, 5,927 Maven Central projects declare a dependency on Apache Struts 2.3.16.3. Any project that uses, either directly or indirectly, one of these Apache Struts releases could benefit from the YouTube video listed in the table. The video not only illustrates how the vulnerability CVE-2017-5638 can be exploited but also shows how to mitigate it.

Table 9-3. Examples of vulnerabilities and their associated YouTube videos

Vulnerability	*Related Video*	*Vulnerable Projects*	*# (Maven Central) Dependencies on Vulnerable Project*
CVE-2017-5638	https://youtu.be/od92kR0MnC4	Apache Struts 2.3.16.3	5927
		Apache Struts 2.3.16.1	5340
		Apache Struts 2.3.8	1173
		Apache Struts 2.3.16	585

It should be noted that the accuracy of these indirect links is dependent on the accuracy of the SBSON-SEVONT alignment (discussed in [105]) and the SEVONT-VIDONT alignment introduced in this chapter. In [105], a precision of 0.87 and a recall of 0.64 for linking our SBSON-SEVONT ontologies was reported.

9.6 Threats to Validity

Our research is introducing a methodology for integrating relevant content from screencast tutorials, which are created by the crowd, with other software security knowledge resources.

However, some threats to validity exist that might affect our reported results and the applicability of our approach.

Construct Validity: We identify three threats that relate to the tools and mechanisms used to obtain our results. The first threat is that our case study relies on our ability to mine facts from both YouTube and the NVD repository to populate our ontologies. A common problem when mining software repositories is that these repositories often contain noise in their data, due to data ambiguity, inconsistencies, or incompleteness. Although studies (e.g., [141]) have shown the presence of incorrect NVD information, this threat is partly mitigated since vulnerabilities published in NVD are curated by security experts. Similarly, the Maven tool ensures that defined project dependencies are fully specified and available in the Maven Central repository, limiting not only ambiguities and inconsistencies at the project build but also at the complete dataset level.

Regarding the knowledge extracted from YouTube, not all vulnerability related videos explicitly mention a vulnerability CVE in their title, description, or content (i.e., image frames text or speech). As our case study has shown, this can significantly reduce the accuracy of our approach to automatically link such videos to the related vulnerabilities. To address this potential limitation, we use the BM25 information retrieval approach [92] to identify NVD vulnerabilities that are most similar to a given video to further improve our alignment between these two resources, since BM25 performs well on shorter textual descriptions. While we are not able to mitigate this threat completely, different approaches and techniques (such as reducing the search space) can be used to improve the linking results.

Another potential threat is the automatic transcription of videos, a process that is prone to errors and potentially can cause CVE IDs not being correctly transcribed. Using a specialized gazetteer list (e.g., WordNet [68]) to detect and correct erroneously transcribed CVE IDs can be considered as future work to improve the accuracy of our approach in terms of identifying these CVE IDs.

The final threat to construct validity is related to the extraction of project dependency information. The work in this chapter does not consider dependency scopes, configurations, and exclusions during project dependency resolution; this can introduce false positives when identifying potentially vulnerable projects based on transitive project dependencies. For future work, we plan to extend our dependency analysis to cover such cases.

Internal validity: One internal threat that potentially can affect our results is the quality of the established links from vulnerabilities to release builds which we used to answer RQ13. SV-AF approach establishes these traceability links with a precision of 0.87 and a recall of 0.64. In addition, we compared SV-AF against a publicly available [129] and a proprietary tool[75] (now open source) [124]; SV-AF's accuracy compared favorably against the free tool and just below the proprietary tool.

External validity: In terms of external threats to validity, a potential threat is that the presented experiments are not generalizable to non-YouTube videos and non-NVD vulnerabilities. This threat can be mitigated by our modeling approach with its different abstraction layers. Our domain-specific ontologies (e.g., SEVONT and VIDONT) contain the core shared concepts and relations common to that particular domain. These domain ontologies can be extended and instantiated to include system level ontologies that support new vulnerability and video repositories. Also, our dataset can be considered incomplete, covering only a limited number of existing videos and vulnerabilities, limiting the generalizability of our results. To mitigate this problem, we plan to extend the datasets as part of our future work to include a larger number of videos and vulnerabilities in our analysis.

9.7 Chapter Summary

The objective of our work in this chapter is to provide an application of our screencast mining approach along with a standardized ontological representation that allows for seamless knowledge integration across abstraction levels and knowledge resource boundaries. Having this knowledge integration not only can provide developers with direct access to vulnerability information described in a screencast content, but also allows to link vulnerability descriptions to relevant screencasts and dependency information. In addition, our approach also allows developers to identify screencasts that demonstrate such attacks and provides developers who are indirectly using vulnerable libraries in their project (e.g., through Maven dependencies) with insights on how to reduce the potential impact of being directly or indirectly exposed to a vulnerability.

[75] The tool in [28] was proprietary at the time the evaluation was performed in our previous work.

We evaluate the flexibility and applicability of our approach to 1) provide bi-directional links between known vulnerabilities and screencasts published on YouTube, and 2) link relevant NVD entries, screencasts, and build dependencies to provide developers and maintainers with valuable insights during vulnerability management and impact (ripple effect) analysis. We performed a case study on 48 videos that describe the exploits of known vulnerabilities and how these vulnerabilities can be fixed. Our evaluation shows that our approach can successfully link relevant vulnerabilities and screencasts with an average precision of 98% and an average recall of 54% when vulnerability identifiers (CVE ID) are explicitly mentioned in the videos. When no direct reference to a CVE ID exists in the screencast, our approach was still able to link video-vulnerability descriptions, with up to 100% of the time relevant links being ranked in the 2nd position of our results set.

10 Conclusion and Future Work

We conclude this thesis with a summary of the research contributions that we provided to validate our thesis hypothesis “*Mining and linking crowd-based screencasts that showcase GUI features of an application is feasible and allows to support a variety of software maintenance and development tasks.*” We also provide future research work.

10.1 Conclusion

While organizations are investing a significant amount of resources in creating workplace environments that accommodate the physical and social needs of their current and future generations of employees (e.g., relaxation areas, gaming rooms, free food), these companies fail to adapt their software processes to meet the information requirements of their changing work population. Deriving strategies to engage and retain this next generation of the workforce will be critical to a business’s bottom line. Generation Y and Z employees will expect from their employers a workplace software ecosystem that includes access to instant messaging, video-on-demand, blogs, wikis, and social networking. These social tools will enable this generation to instantly connect, engage, and collaborate with cohorts in ways that are natural to them, leading to better productivity across the enterprise. For example, while current best software practices provide system documentation that supports different system stakeholders, this documentation might not adhere to the changing communication and learning needs of future generations of employees.

Our research presents an approach that allows us to incorporate mining and linking relevant content from multimedia resources (screencasts), allowing the provision of supplementary media types and allowing this multimedia content to become seamlessly integrated with other software artifacts. The inclusion of such crowd-based multimedia documentation in different formats of text, image, audio, and video into existing software documentation is one aspect for corporations to provide all generations of employees with a work and learning environment they are accustomed to and prefer.

However, such integration of multi-media content in existing software processes also faces several challenges (e.g., quality and quantity of multimedia documents, see Chapter 2). Incentives are needed for people to produce and publish high-quality videos and screencasts that provide content useful to software developers. Reputation and financial compensation incentives are one way to encourage the creation of quality screencasts by the crowd. Beyond content creation and design, the quality of videos concerning their image quality, languages and dialects used by narrators, and the length of the videos are also essential aspects when analyzing multi-media content.

To gain a better insight on the usage of multimedia documentation and specifically tutorial screencasts in the software engineering domain (see Chapter 5), we conducted a survey to which we received 99 responses. We analyzed the responses and found that "YouTube" and "Videos" are more preferred by less experienced software engineers, while more experienced ones prefer to use "Question and answer sites" such as "Stack Overflow" as an information source. Most of our survey participants use tutorial screencasts for understanding and learning new concepts. Also, non-technical software engineering-related purposes (e.g., getting familiar with GUI features of an application) are other reasons why most of our respondents watch tutorial videos. We also found that the content and title of videos are the main information sources for the respondents to use when selecting the right tutorial to watch. Considering GUI-based tutorial screencasts as a software artifact, we found that users try to trace down the features shown in the screencast to the corresponding source code artifact.

These findings support the need for accommodating software engineers with different backgrounds and expertise and, therefore, the proposal of a methodology of mining and linking crowd-based screen captured videos describing the graphical workflow of a software feature. Our methodology makes use of the content of screencasts (i.e., spoken and OCR text). We then presented our proposed methodology and three applications of it, namely extracting use-case scenarios from speech, feature location using screencasts, and a Semantic Web-based approach of linking vulnerability exploitation screencasts to their NVD vulnerability descriptions. We conducted case studies to evaluate these applications and therefore our thesis hypothesis.

We found that our approach of mining speech of screencasts can extract use-case steps from spoken text and at the same time summarize it. Therefore, such extracted documentation can be used as a supplementary source of information for software engineers. Similar to our survey findings, we found that the content of videos (in this case study the content of spoken text) is a better relevancy indicator than what other search engines use (e.g., title, description, tags).

Evaluation of our feature location approach revealed interesting information about how the level of noise and technicality of the presented information in a screencast can affect the performance of locating source code using screencasts. Accordingly, rebalancing the speech and GUI data in use-case scenarios that contain less noise turned out to outperform when using original data and improve the results of locating source code relevant to both the technical implementation of a specific scenario or backend development of it and high-level implementation of a scenario or its frontend development. While rebalancing noisy screencast data improves the results for the high-level implementation of a scenario or its frontend development, the performance of our approach is less consistent for the technical implementation of the scenario. Also, we found that speech data, if clear and concise, is an important information source that can be used for linking screencasts to source code. Furthermore, we found that using GUI-based screencasts for locating source code relevant to the frontend development of a project outperforms when locating backend-related source code. Lastly, we compared the performance of our approach with a guided search approach and found that with a similar accuracy to guided search, our approach saves a significant amount of time when locating relevant source code directories.

We took a step further and leveraged a Semantic Web-based approach in our methodology to establish 1) bi-directional links between known vulnerability descriptions on NVD and their relevant vulnerability exploitation screencasts on YouTube, and 2) indirect links between vulnerability descriptions, screencasts, and their relevant build dependencies. We evaluated the approach and found that when the CVE ID is present the approach is highly performant, while when no CVE ID is present in a screencast further filtering based on the product and CWE ID is needed to reduce the search space and attain a high accuracy.

Based on our evaluations, the mining and linking of crowd-based screencasts that showcase GUI features of an application is feasible and allows to support a variety of software maintenance and development tasks (e.g., extracting use-case scenarios and establishing traceability links). This approach provides the basis for building a software ecosystem that enables software engineers with different backgrounds and documentation preferences to easily interact with the system and meet their information needs.

10.2 Recommendations

As a result of our findings and based on our survey results (Chapter 5), we are motivated to present general guidelines that, if applied, can further improve the automated mining of screencasts and linking features demonstrated in them to their corresponding source code implementation. YouTube already has a dedicated platform[76] to provide general guidelines for video creators in different genres (education, beauty, music, gaming, etc.) to boost their channel subscriptions and keep their audience engaged and satisfied with the content presentation, quality, and design. However, the provided guidelines to build educational channels[77] that are close to tutorial videos that we studied in this thesis are not specific to any field, which in our case is software engineering-related technical videos. Also, video production guidelines[78] lack guidance for creating high quality **screencasts,** which requires screen recording, on YouTube. Also, the provided guidelines for YouTube search and discovery[79] do not consider the content (of GUI and speech) in (screen captured) videos since YouTube's search works based on metadata, title, and thumbnails.

Our case studies' results (Chapters 7, 8, and 9) confirm other researchers' findings [9] about best practices in creating software engineering-related tutorial screencasts. The performance of our automated approach can be improved if developers and video creators follow these best practices when creating a screencast that showcases GUI features of an application (items that are marked by an * are also presented in YouTube Creator Academy):

1) *Provide a subtitle. Screencast creators should provide a (English) transcript of their video and its content (even if it is in a language other than English[80]), eliminating the dependency on automated transcription services, which are still prone to errors.

[76] https://creatoracademy.youtube.com/page/home
[77] https://creatoracademy.youtube.com/page/course/educational-channel?hl=en
[78] https://creatoracademy.youtube.com/page/home
[79] https://creatoracademy.youtube.com/page/lesson/discovery?hl=en
[80] https://creatoracademy.youtube.com/page/lesson/global-format#strategies-zippy-link-3

2) In cases when no video subtitle is provided by the video creator, the screencasts should be presented using high-quality speech that is understandable, has less noise, and can be transcribed with high accuracy.

3) Recording high definition (HD) videos that have a readable text size [9], which can improve the accuracy of the OCR's output.

4) Reducing noise in videos by showing only the features that are related to the topic that is being presented in the video and by minimizing the amount of time spent on showing non-relevant subject on the screen. This can be done by providing links to other screencasts that cover other topics [9].

5) *Providing complementary documents [9] such as written documents or slides that can further improve the linking results.

6) *Annotating or tagging screencasts using keywords that can help to clearly distinguish one video that covers a use-case scenario from a video that covers another use-case scenario [9].

7) How-to screencasts should include information about the version of the software application whose features are being demonstrated to improve the location of the correct source code version.

Also, our findings confirm other software traceability researchers' findings [52] about programming guidelines and conventions that should be followed by developers to enable better source code localization:

8) Modifying the projects' naming conventions to reflect the high-level purpose or feature that a source code artifact (method, identifier, etc.) is implementing.

9) Including comments in the source code that describe what high-level features of the project are implemented by this source code.

10) Applying separation of concerns and designing modular code via the use of design patterns [142].

10.3 Future Work

In this section, we provide future directions that can help improve each part of our work including our survey on the adoption of crowd-based software engineering tutorial screencasts (Chapter 5), mining crowd-based speech documentation (Chapter 7), feature location using crowd-based screencasts (Chapter 8), and our Semantic Web-based approach for integrating screencasts with security advisories (Chapter 9).

Adoption of crowd-based software engineering tutorial screencasts: As part of our future work, to reduce the effects of threats to validity, we plan to increase the number of survey participants coming from different communities, especially the open source community, and with a more diverse expertise and ages to gain clearer insights on the usage of how-to screencasts by different users. We also plan to conduct additional user studies that will involve follow-up interviews and additional application scenarios covering different types of software applications (e.g., web-based, desktop applications, etc.) to better understand how GUI-based screencasts that show how to use application features are used as software artifacts by different types of software engineers.

Mining crowd-based speech documentation: As part of our future work, one can evaluate different statistical models to extract abstract topics and improve our sentence segmentation using other techniques, such as language models. At a higher level, applying the methodology on different software applications and use-case scenarios and conducting a user study on the usage of the recovered use-case scenarios from speech can be done as future work.

Feature location using crowd-based screencasts: Future work should extend the data sets by including additional screencasts involving different scenarios and software applications. Also, while our work established that screencasts contain enough textual information, future work should extend our approach with other text retrieval-based approaches for feature location. Furthermore, we plan to conduct a user study on our feature location results using screencasts to better understand the benefits of our approach and receive feedback on what needs to be improved.

Semantic Web-based approach for integrating screencasts with security advisories: As part of our future work, we plan to make the filtering process (introduced in the case study) to reduce the search space an integrated part of our linking approach by extending our current knowledge model to include a CWE ontology and link to DBpedia. Having such a unified knowledge representation will allow us to infer additional knowledge and further restrict our search space during the linking process of videos which do not contain a CVE ID. We also plan to conduct another empirical study to improve the generalizability of our approach by covering different vulnerability and video repositories. In addition, having a larger dataset would also allow us to classify vulnerability related videos based on their popularity and content. We also consider extending our modeling approach to integrate videos and their content with other software artifacts such as blogs and Q/A forums (e.g., Stack Overflow) to derive new application scenarios for our modeling approach.

Other future work: In addition to the future directions that were discussed for each application of the work presented in this thesis, future work should consider building and implementing other applications of the proposed methodology as well as conducting a user study to evaluate the applicability of the methodology in different contexts. Also, developing tools and plugins that enable advanced content-based screencast search and live traceability links between code, textual documentation, and videos as well as maintaining such links as code (or other artifacts) evolves should be considered. Furthermore, exploiting other information sources from screencasts (e.g., sequence of events, see Section 6.1) to build tools that perform more advanced video content search and/or linking these sequences of events to their corresponding call graph can be considered as future work. Such tools can bring the proposed applications into practice for organizations and software engineers with different backgrounds and expertise and speed up the adoption of crowd-based multimedia documentation and, more specifically, screencasts.

References

[1] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, “The (R) Evolution of social media in software engineering,” *Proc. Futur. Softw. Eng. - FOSE 2014*, pp. 100–116, 2014.

[2] L. MacLeod, M.-A. Storey, and A. Bergen, “Code, Camera, Action: How Software Developers Document and Share Program Knowledge Using YouTube,” *2015 IEEE 23rd Int. Conf. Progr. Compr.*, 2015.

[3] L. Ponzanelli *et al.*, “Too long; didn’t watch!: extracting relevant fragments from software development video tutorials,” in *Proceedings of the 38th International Conference on Software Engineering - ICSE ’16*, 2016, pp. 261–272.

[4] O. Barzilay, C. Treude, and A. Zagalsky, “Facilitating Crowd Sourced Software Engineering via Stack Overflow,” in *Finding Source Code on the Web for Remix and Reuse*, New York: Springer New York, 2013, pp. 1–19.

[5] S. Black, R. Harrison, and M. Baldwin, “A Survey of Social Media Use in Software Systems Development,” *Proc. 1st Work. Web 2.0 Softw. Eng.*, pp. 1–5, 2010.

[6] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, “Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow,” *Georg. Tech Tech. Rep.*, 2012.

[7] M. McLure Wasko and S. Faraj, “‘It is what one does’: why people participate and help others in electronic communities of practice,” *J. Strateg. Inf. Syst.*, vol. 9, no. 2–3, pp. 155–173, Sep. 2000.

[8] M.-A. Storey, “Selecting research methods for studying a participatory culture in software development,” in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering - EASE ’15*, 2015, pp. 1–5.

[9] L. MacLeod, A. Bergen, and M.-A. Storey, “Documenting and sharing software knowledge using screencasts,” *Empir. Softw. Eng.*, vol. 22, no. 3, pp. 1478–1507, Jun. 2017.

[10] E. E. Eghan, P. Moslehi, J. Rilling, and B. Adams, “The missing link – A semantic web based approach for integrating screencasts with security advisories,” *Inf. Softw. Technol.*, vol. 117, p. 106197, Jan. 2020.

[11] A. Ahmad, C. Feng, S. Ge, and A. Yousif, “A survey on mining stack overflow: question and answering (Q&amp;A) community,” *Data Technol. Appl.*, vol. 52, no. 2, pp. 190–247, Apr. 2018.

[12] L. Ponzanelli *et al.*, “Automatic Identification and Classification of Software Development Video Tutorial Fragments,” *IEEE Trans. Softw. Eng.*, vol. 45, pp. 464–488, 2019.

[13] K. Khandwala and P. J. Guo, “Codemotion: Expanding the Design Space of Learner Interactions with Computer Programming Tutorial Videos,” in *Proceedings of the Fifth Annual ACM Conference on Learning at Scale - L@S ’18*, 2018, pp. 1–10.

[14] P. Moslehi, B. Adams, and J. Rilling, “Feature Location using Crowd-based Screencasts,” in *Proceedings of the 15th IEEE Working Conference on Mining Software Repositories (MSR)*, 2018, pp. 192–202.

[15] L. Bao, J. Li, Z. Xing, X. Wang, X. Xia, and B. Zhou, “Extracting and analyzing time-series HCI data from screen-captured task videos,” *Empir. Softw. Eng.*, vol. 22, no. 1, pp. 134–174, Feb. 2017.

[16] L. Bao, Z. Xing, X. Xia, and D. Lo, “VT-Revolution: Interactive Programming Video Tutorial Authoring and Watching System,” *IEEE Trans. Softw. Eng.*, vol. 45, pp. 823–838, 2019.

[17] L. Bao, Z. Xing, X. Wang, and B. Zhou, “Tracking and Analyzing Cross-Cutting Activities in Developers’ Daily Work (N),” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 277–282.

[18] P. Moslehi, B. Adams, and J. Rilling, “On mining crowd-based speech documentation,” in *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR ’16*, 2016, pp. 259–268.

[19] A. Singh, “Challenges and Issues of Generation Z,” *IOSR J. Bus. Manag.*, vol. 16, no. 7, pp. 59–63, 2014.

[20] C. Seemiller and M. Grace, “Generation Z: Educating and Engaging the Next Generation of Students,” *About Campus*, vol. 22, no. 3, pp. 21–26, 2017.

[21] A. Turner, “Generation Z: Technology and Social Interest,” *J. Individ. Psychol.*, 2015.

[22] S. P. Eisner, “Managing generation Y,” *IEEE Eng. Manag. Rev.*, vol. 39, no. 2, pp. 6–18, 2011.

[23] H. J. Krahn and N. L. Galambos, “Work values and beliefs of ‘Generation X’ and ‘Generation Y,’” *J. Youth Stud.*, 2014.

[24] T. Reisenwitz, “Differences in Generation X and Generation Y: Implications for the Organization and Marketers,” *Mark. Manag. J.*, 2009.

[25] J. Krug, “Understanding generation x,” *J. Manag. Eng.*, 1998.

[26] S. Lissitsa and O. Kol, “Generation X vs. Generation Y - A decade of online shopping,” *J. Retail. Consum. Serv.*, 2016.

[27] M. McCrindle and E. Wolfinger, *The ABC of XYZ : understanding the global generations / Mark McCrindle with Emily Wolfinger*. UNSW Press Sydney, 2009.

[28] D. R. Garrison and H. Kanuka, “Blended learning: Uncovering its transformative potential in higher education,” *Internet High. Educ.*, vol. 7, no. 2, pp. 95–105, Apr. 2004.

[29] S. Subramanian, L. Inozemtseva, and R. Holmes, “Live API documentation,” in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, 2014, pp. 643–652.

[30] W. Sugar, A. Brown, and K. Luterbach, “Uncovering Common Elements and Instructional Strategies,” vol. 11, no. 3, 2010.

[31] D. M. Blei, “Probabilistic topic models,” in *Communications of the ACM*, 2012, vol. 55, no. 4, pp. 77–84.

[32] T.-H. Chen, S. W. Thomas, and A. E. Hassan, “A survey on the use of topic models when mining software repositories,” *Empir. Softw. Eng.*, vol. 21, no. 5, pp. 1843–1919, Oct. 2016.

[33] A. Kuhn, P. Loretan, and O. Nierstrasz, “Consistent Layout for Thematic Software Maps,” Sep. 2012.

[34] A. Kuhn, S. Ducasse, and T. Gîrba, “Semantic clustering: Identifying topics in source code,” *Inf. Softw. Technol.*, vol. 49, no. 3, pp. 230–243, Mar. 2007.

[35] B. Bassett and N. A. Kraft, “Structural information based term weighting in text retrieval for feature location,” in *2013 21st International Conference on Program Comprehension (ICPC)*, 2013, pp. 133–141.

[36] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya, “A theory of aspects as latent topics,” in *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, 2008, vol. 43, no. 10, pp. 543–562.

[37] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, “Duplicate bug report detection with a combination of information retrieval and topic modeling,” in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, 2012, pp. 70–79.

[38] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, “Validating the Use of Topic Models for Software Evolution,” in *2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, 2010, pp. 55–64.

[39] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, “Software traceability with topic modeling,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE ’10*, 2010, vol. 1, pp. 95–104.

[40] N. Ali, A. Sabane, Y. Gueheneuc, and G. Antoniol, “Improving Bug Location Using Binary Class Relationships,” in *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*, 2012, pp. 174–183.

[41] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby, “A search engine for finding highly relevant applications,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, 2010, vol. 1, pp. 475–484.

[42] S. K. Bajracharya and C. V. Lopes, “Analyzing and mining a code search engine usage log,” *Empir. Softw. Eng.*, vol. 17, no. 4–5, pp. 424–466, Aug. 2012.

[43] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[44] H. M. Wallach, “Topic Modeling: Beyond Bag-of-words,” in *Proceedings of the 23rd International Conference on Machine Learning*, 2006, pp. 977–984.

[45] T. Hofmann and Thomas, “Unsupervised Learning by Probabilistic Latent Semantic Analysis,” *Mach. Learn.*, vol. 42, no. 1/2, pp. 177–196, 2001.

[46] D. M. Blei, M. I. Jordan, T. L. Griffiths, and J. B. Tenenbaum, “Hierarchical Topic Models and the Nested Chinese Restaurant Process,” in *Proceedings of the 16th International Conference on Neural Information Processing Systems*, 2003, pp. 17–24.

[47] J. D. Mcauliffe and D. M. Blei, “Supervised Topic Models,” in *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. Curran Associates, Inc., 2008, pp. 121–128.

[48] D. Ramage, D. Hall, R. Nallapati, and C. D. Manning, “Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora,” in *EMNLP 2009 - Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: A Meeting of SIGDAT, a Special Interest Group of ACL, Held in Conjunction with ACL-IJCNLP 2009*, 2009, pp. 248–256.

[49] J. Cleland-Huang, O. C. Z. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, “Software traceability: trends and future directions,” in *Proceedings of the on Future of Software Engineering - FOSE 2014*, 2014, pp. 55–69.

[50] J. Cleland-Huang, P. Mader, M. Mirakhorli, and S. Amornborvornwong, “Breaking the big-bang practice of traceability: Pushing timely trace recommendations to project stakeholders,” in *2012 20th IEEE International Requirements Engineering Conference (RE)*, 2012, pp. 231–240.

[51] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, “Feature location in source code: A taxonomy and survey,” *J. Softw. Evol. Process*, vol. 25, no. 1, pp. 53–95, 2013.

[52] O. Gotel *et al.*, “The Grand Challenge of Traceability (v1.0),” in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. London: Springer London, 2012, pp. 343–409.

[53] J. E. Gaffney Jr., “Metrics in Software Quality Assurance,” in *Proceedings of the ACM ’81 Conference*, 1981, pp. 126–130.

[54] H. Kagdi and J. I. Maletic, “Software Repositories : A Source for Traceability Links,” *TEFSE/GCT 2007 - 4th Int. Work. Traceability Emerg. Forms Softw. Eng.*, no. APRIL 2002, pp. 32–39, 2007.

[55] H. Kagdi, J. I. Maletic, and B. Sharif, “Mining software repositories for traceability links,” in *15th IEEE International Conference on Program Comprehension (ICPC ’07)*, 2007, pp. 145–154.

[56] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web,” *Sci. Am.*, vol. 284, no. 5, pp. 34–43, May 2001.

[57] D. L. McGuinness and F. Van Harmelen, “Owl web ontology language overview,” *W3C Recomm. 10.2004-03*, vol. 2004, no. February, pp. 1–12, 2004.

[58] C. J. H. Mann, “The Description Logic Handbook – Theory, Implementation and Applications,” *Kybernetes*, vol. 32, no. 9/10, p. k.2003.06732iae.006, Dec. 2003.

[59] B. DuCharme, *Learning SPARQL*, 2n Edition. O’Reilly Media, 2011.

[60] B. Henderson-Sellers, “Bridging metamodels and ontologies in software engineering,” *J. Syst. Softw.*, vol. 84, no. 2, pp. 301–313, Feb. 2011.

[61] R. Witte, Y. Zhang, and J. Rilling, “Empowering software maintainers with semantic web technologies,” *Eur. Conf. Semant. Web Res. Appl.*, pp. 37–52, 2007.

[62] C. Atkinson, M. Gutheil, and K. Kiko, “On the relationship of ontologies and models,” *Proc. 2nd Work. MetaModelling Ontol. WoMM06 LNI P96 Gesellschaft fur Inform. Bonn*, pp. 47–60, 2006.

[63] H. C. Jiau and F.-P. Yang, “Facing up to the inequality of crowdsourced API documentation,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 37, no. 1, pp. 1–9, Jan. 2012.

[64] J. C. Campbell, C. Zhang, Z. Xu, A. Hindle, and J. Miller, “Deficient documentation detection: A methodology to locate deficient project documentation using topic analysis,” *IEEE Int. Work. Conf. Min. Softw. Repos.*, pp. 57–60, 2013.

[65] R. Pham, L. Singer, O. Liskin, F. F. Filho, and K. Schneider, “Creating a shared understanding of testing culture on a social coding site,” in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 112–121.

[66] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, “What makes a good code example?: A

study of programming Q&A in StackOverflow," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, pp. 25–34.

[67] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *11th Working Conference on Reverse Engineering*, 2004, pp. 214–223.

[68] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *J. Am. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, 1990.

[69] P. van der Spek, S. Klusener, and P. van de Laar, "Towards Recovering Architectural Concepts Using Latent Semantic Indexing," in *2008 12th European Conference on Software Maintenance and Reengineering*, 2008, pp. 253–257.

[70] J. Eddy, Brian P. and Kraft, Nicholas A. and Gray, "Impact of structural weighting on a latent Dirichlet allocation–based feature location technique," *J. Softw. Evol. Process*, vol. 30, no. 1, p. e1892, 2018.

[71] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent Dirichlet allocation," *Inf. Softw. Technol.*, vol. 52, no. 9, pp. 972–990, 2010.

[72] L. Bao, J. Li, Z. Xing, X. Wang, and B. Zhou, "Reverse engineering time-series interaction data from screen-captured videos," *2015 IEEE 22nd Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2015 - Proc.*, pp. 399–408, Apr. 2015.

[73] S. Yadid and E. Yahav, "Extracting code from programming tutorial videos," in *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software - Onward! 2016*, 2016, pp. 98–111.

[74] J. Ott, A. Atchison, P. Harnack, A. Bergh, and E. Linstead, "A Deep Learning Approach to Identifying Source Code in Images and Video," *Int. Conf. Min. Softw. Repos.*, pp. 376–386, 2018.

[75] J. Escobar-Avila, E. Parra, and S. Haiduc, "Text Retrieval-Based Tagging of Software Engineering Video Tutorials," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 341–343.

[76] E. Parra, J. Escobar-Avila, and S. Haiduc, "Automatic tag recommendation for software development video tutorials," in *Proceedings of the 26th Conference on Program Comprehension - ICPC '18*, 2018, pp. 222–232.

[77] E. Poche, N. Jha, G. Williams, J. Staten, M. Vesper, and A. Mahmoud, "Analyzing User Comments on YouTube Coding Tutorial Videos," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, 2017, pp. 196–206.

[78] M. Ellmann, A. Oeser, D. Fucci, and W. Maalej, "Find, Understand, and Extend Development Screencasts on YouTube," *Proc. 3rd ACM SIGSOFT Int. Work. Softw. Anal. - SWAN 2017*, pp. 1–7, 2017.

[79] J. Wells, R. M. Barry, and A. Spence, "Using Video Tutorials as a Carrot-and-Stick Approach to Learning," *IEEE Trans. Educ.*, vol. 55, no. 4, pp. 453–458, Nov. 2012.

[80] S. Mohorovičič, "Creation and use of screencasts in higher education," *MIPRO 2012 - 35th Int. Conv. Inf. Commun. Technol. Electron. Microelectron. - Proc.*, pp. 1293–1298, 2012.

[81] M. B. Miles, A. M. Huberman, and J. Saldana, *Qualitative Data Analysis: A Methods Sourcebook.* SAGE Publications, 2018.

[82] A. Pappu and A. Stent, "Automatic formatted transcripts for videos," in *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2015, vol. 2015-Janua, pp. 2514–2518.

[83] R. Brunelli and T. Poggio, "Face recognition: features versus templates," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 10, pp. 1042–1052, Oct. 1993.

[84] M. S. Nixon and A. S. Aguado, "Chapter 5 - High-level feature extraction: fixed shape matching," in *Feature Extraction and Image Processing for Computer Vision (Third edition)*, Third edit., M. S. Nixon and A. S. Aguado, Eds. Oxford: Academic Press, 2012, pp. 217–291.

[85] M. S. Nixon and A. S. Aguado, "Chapter 7 - Object description," in *Feature Extraction and Image Processing for Computer Vision (Third edition)*, Third edit., M. S. Nixon and A. S. Aguado, Eds. Oxford: Academic Press, 2012, pp. 343–397.

[86] M. Cheriet, N. Kharma, C. Liu, and C. Suen, *Character Recognition Systems: A Guide for Students and Practitioners*. Wiley-Interscience, 2007.

[87] S. Takahashi and T. Morimoto, "N-gram Language Model Based on Multi-Word Expressions in Web Documents for Speech Recognition and Closed-Captioning," in *2012 International Conference on Asian Language Processing*, 2012, pp. 225–228.

[88] T. Oba, T. Hori, and A. Nakamura, "Sentence Boundary Detection Using Sequential Dependency Analysis Combined with CRF-based Chunking," *Corpus*, vol. 3, pp. 1153–1156, 2006.

[89] T. Oba, T. Hori, and A. Nakamura, "Improved sequential dependency analysis integrating labeling-based sentence boundary detection," *IEICE Trans. Inf. Syst.*, vol. E93-D, no. 5, pp. 1272–1281, 2010.

[90] Michael W. Berry, Ed., *Survey of Text Mining*. Springer New York, 2004.

[91] K. Adrian, E. David, L. Peter, and N. Oscar, "Software Cartography: thematic software visualization with consistent layout," *J. Softw. Maint. Evol. Res. Pract.*, vol. 22, no. 3, pp. 191–210.

[92] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008.

[93] X. Wei and W. B. Croft, "LDA-Based Document Models for Ad-hoc Retrieval," 2006.

[94] S. W. Thomas, "Mining Unstructured Software Repositories Using IR Models," Queen's University, 2012.

[95] D. Jurafsky and J. H. Martin, *Speech and Language Processing (2nd Edition)*. USA: Prentice-Hall, Inc., 2009.

[96] X. Wang, A. McCallum, and X. Wei, "Topical N-grams: Phrase and topic discovery, with an application to information retrieval," *Proc. - IEEE Int. Conf. Data Mining, ICDM*, pp. 697–702, 2007.

[97] I. Keivanloo, "Source Code Similarity and Clone Search," 2013.

[98] M. Baroni, S. Bernardini, A. Ferraresi, and E. Zanchetta, "The WaCky wide web: a collection of very large linguistically processed web-crawled corpora," *Lang. Resour. Eval.*, vol. 43, no. 3, pp. 209–226, 2009.

[99] W. D. Gray, *Integrated Models of Cognitive Systems (Advances in Cognitive Models and Architectures)*. New York, NY, USA: Oxford University Press, Inc., 2007.

[100] G. Pedrosa, M. Pita, P. Bicalho, A. Lacerda, and G. L. Pappa, “Topic Modeling for Short Texts with Co-occurrence Frequency-Based Expansion,” *Proc. - 2016 5th Brazilian Conf. Intell. Syst. BRACIS 2016*, pp. 277–282, 2017.

[101] X. Cheng, X. Yan, Y. Lan, and J. Guo, “BTM: Topic modeling over short texts,” *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 12, pp. 2928–2941, 2014.

[102] C. Li, H. Wang, Z. Zhang, A. Sun, and Z. Ma, “Topic Modeling for Short Texts with Auxiliary Word Embeddings,” in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval - SIGIR ’16*, 2016, pp. 165–174.

[103] P. T. Devanbu and S. Stubblebine, “Software engineering for security: a roadmap,” in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 227–239.

[104] A. E. Hassan, “The road ahead for Mining Software Repositories,” in *2008 Frontiers of Software Maintenance*, 2008, pp. 48–57.

[105] S. S. Alqahtani, E. E. Eghan, and J. Rilling, “SV-AF - A Security Vulnerability Analysis Framework,” in *IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016, pp. 219–229.

[106] J. Williams and A. Dabirsiaghi, “The unfortunate reality of insecure libraries,” pp. 1–26, 2012.

[107] I. Sonatype, *Maven: The Definitive Guide*. O’Reilly, 2008.

[108] M. Würsch, G. Ghezzi, M. Hert, G. Reif, and H. C. Gall, “SEON: a pyramid of ontologies for software evolution and its applications,” *Computing*, vol. 94, no. 11, pp. 857–885, Nov. 2012.

[109] A. Kimmig, S. Bach, M. Broecheler, B. Huang, and L. Getoor, “A short introduction to probabilistic soft logic,” in *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*, 2012, pp. 1–4.

[110] L. Yu, J. Zhou, Y. Yi, P. Li, and Q. Wang, “Ontology Model-Based Static Analysis on Java Programs,” in *2008 32nd Annual IEEE International Computer Software and Applications Conference*, 2008, pp. 92–99.

[111] H.-J. Happel, A. Korthaus, S. Seedorf, and P. Tomczyk, “KOntoR: An Ontology-enabled Approach to Software Reuse,” in *In: Proc. Of The 18Th Int. Conf. On Software Engineering And Knowledge Engineering*, 2006.

[112] J. Tappolet, C. Kiefer, and A. Bernstein, “Semantic web enabled software analysis,” *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 8, no. 2–3, pp. 225–240, Jul. 2010.

[113] H. C. Jiau and F.-P. Yang, “Facing up to the inequality of crowdsourced API documentation,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 37, no. 1, p. 1, Jan. 2012.

[114] O. Barzilay, C. Treude, and A. Zagalsky, “Facilitating Crowd Sourced Software Engineering via Stack Overflow,” in *Finding Source Code on the Web for …*, New York: Springer New York, 2013, pp. 1–19.

[115] S. Subramanian, L. Inozemtseva, and R. Holmes, “Live API documentation,” in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, 2014, pp. 643–652.

[116] L. Bao, Z. Xing, X. Wang, and B. Zhou, “Tracking and Analyzing Cross-Cutting Activities in Developers’ Daily Work,” in *30th IEEE/ACM International Conference on*

Automated Software Engineering (ASE 2015), 2015, pp. 277–282.

[117] L. Ponzanelli *et al.*, “Automatic Identification and Classification of Software Development Video Tutorial Fragments,” *IEEE Trans. Softw. Eng.*, p. 1, 2017.

[118] S. Yadid and E. Yahav, “Extracting code from programming tutorial videos,” in *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software - Onward! 2016*, 2016, pp. 98–111.

[119] L. MacLeod, A. Bergen, and M.-A. Storey, “Documenting and sharing software knowledge using screencasts,” *Empir. Softw. Eng.*, vol. 22, no. 3, pp. 1478–1507, Jun. 2017.

[120] M. Hirzel, D. Von Dincklage, A. Diwan, and M. Hind, “Fast online pointer analysis,” *ACM Trans. Program. Lang. Syst.*, vol. 29, no. 2, pp. 11–66, Apr. 2007.

[121] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner, “Bunch: A clustering tool for the recovery and maintenance of software system structures,” in *Software Maintenance, 1999.(ICSM’99) Proceedings. IEEE International Conference on*, 1999, pp. 50–59.

[122] J.-D. Choi, M. Burke, and P. Carini, “Efficient flow-sensitive interprocedural computation of pointer-induced aliases and side effects,” in *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL ’93*, 1993, pp. 232–245.

[123] N. Rutar, C. B. Almazan, and J. S. Foster, “A Comparison of Bug Finding Tools for Java,” in *15th International Symposium on Software Reliability Engineering*, 2004, pp. 245–256.

[124] H. Plate, S. E. Ponta, and A. Sabetta, “Impact assessment for vulnerabilities in open-source software libraries,” *2015 IEEE 31st Int. Conf. Softw. Maint. Evol. ICSME 2015 - Proc.*, pp. 411–420, 2015.

[125] M. Cadariu, E. Bouwers, J. Visser, and A. Van Deursen, “Tracking known security vulnerabilities in proprietary software systems,” *2015 IEEE 22nd Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2015 - Proc.*, pp. 516–519, 2015.

[126] S. E. Ponta, H. Plate, and A. Sabetta, “Beyond Metadata: Code-Centric and Usage-Based Analysis of Known Vulnerabilities in Open-Source Software,” in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 449–460.

[127] A. Decan, T. Mens, and E. Constantinou, “On the impact of security vulnerabilities in the npm package dependency network,” in *Proceedings of the 15th International Conference on Mining Software Repositories - MSR ’18*, 2018, pp. 181–191.

[128] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, “Vulnerable open source dependencies,” in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM ’18*, 2018, pp. 1–10.

[129] J. Long, S. Springett, and W. Stranathan, “OWASP Dependency Check,” 2015. .

[130] S. S. Alqahtani, E. E. Eghan, and J. Rilling, “Tracing known security vulnerabilities in software repositories - A Semantic Web enabled modeling approach,” *Sci. Comput. Program.*, vol. 121, pp. 153–175, Feb. 2016.

[131] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, “Do developers update their library dependencies?,” *Empir. Softw. Eng.*, vol. 23, no. 1, pp. 384–417, Feb. 2018.

[132] E. E. Eghan, S. S. Alqahtani, C. Forbes, and J. Rilling, “API trustworthiness: an

ontological approach for software library adoption," *Softw. Qual. J.*, pp. 1–46, 2019.

[133] D. Hyland-Wood, D. Carrington, and S. Kaplan, "Toward a Software Maintenance Methodology using Semantic Web Techniques," in *2006 Second International IEEE Workshop on Software Evolvability (SE'06)*, 2006, pp. 23–30.

[134] M. F. Bertoa, A. Vallecillo, and F. García, "An Ontology for Software Measurement," in *Ontologies for Software Engineering and Software Technology*, Springer Berlin Heidelberg, 2006, pp. 175–196.

[135] J. Dietrich and C. Elgar, "A Formal Description of Design Patterns Using OWL," in *Australian Software Engineering Conference*, 2005, pp. 243–250.

[136] C. Kiefer, A. Bernstein, and J. Tappolet, "Mining Software Repositories with iSPAROL and a Software Evolution Ontology," in *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*, 2007, pp. 10–10.

[137] J. Tappolet, "Semantics-aware software project repositories," in *Proceedings of the European Semantic Web Conference, Ph.D. Symposium*, 2008, vol. 8, p. 8.

[138] A. Iqbal, G. Tummarello, M. Hausenblas, and O.-E. Ureche, "LD2SD: linked data driven software development," in *International Conference on Software Engineering & Knowledge Engineering*, 2009.

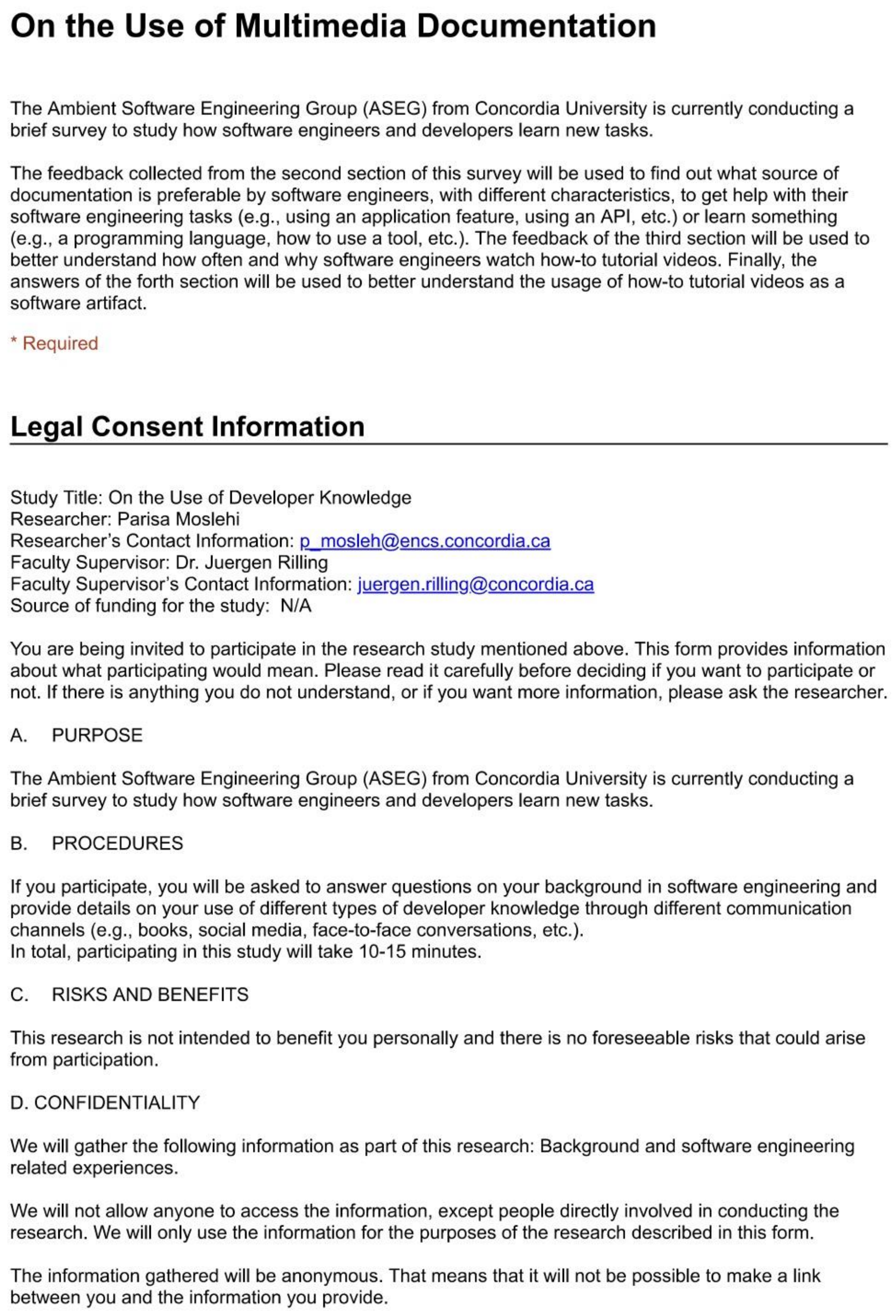
[139] S. S. Alqahtani, E. E. Eghan, and J. Rilling, "Recovering Semantic Traceability Links between APIs and Security Vulnerabilities: An Ontological Modeling Approach," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2017, pp. 80–91.

[140] I. Keivanloo, *Source Code Similarity and Clone Search*. Concordia University, 2013.

[141] V. H. Nguyen and F. Massacci, "The (un)reliability of NVD vulnerable versions data," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13*, 2013, pp. 493–498.

[142] R. J. Leach, *Introduction to Software Engineering*. Boca Raton, FL, USA: CRC Press, Inc., 2000.

Appendix



On the Use of Multimedia Documentation

The Ambient Software Engineering Group (ASEG) from Concordia University is currently conducting a brief survey to study how software engineers and developers learn new tasks.

The feedback collected from the second section of this survey will be used to find out what source of documentation is preferable by software engineers, with different characteristics, to get help with their software engineering tasks (e.g., using an application feature, using an API, etc.) or learn something (e.g., a programming language, how to use a tool, etc.). The feedback of the third section will be used to better understand how often and why software engineers watch how-to tutorial videos. Finally, the answers of the forth section will be used to better understand the usage of how-to tutorial videos as a software artifact.

* Required

Legal Consent Information

Study Title: On the Use of Developer Knowledge
Researcher: Parisa Moslehi
Researcher's Contact Information: p_mosleh@encs.concordia.ca
Faculty Supervisor: Dr. Juergen Rilling
Faculty Supervisor's Contact Information: juergen.rilling@concordia.ca
Source of funding for the study: N/A

You are being invited to participate in the research study mentioned above. This form provides information about what participating would mean. Please read it carefully before deciding if you want to participate or not. If there is anything you do not understand, or if you want more information, please ask the researcher.

A. PURPOSE

The Ambient Software Engineering Group (ASEG) from Concordia University is currently conducting a brief survey to study how software engineers and developers learn new tasks.

B. PROCEDURES

If you participate, you will be asked to answer questions on your background in software engineering and provide details on your use of different types of developer knowledge through different communication channels (e.g., books, social media, face-to-face conversations, etc.).
In total, participating in this study will take 10-15 minutes.

C. RISKS AND BENEFITS

This research is not intended to benefit you personally and there is no foreseeable risks that could arise from participation.

D. CONFIDENTIALITY

We will gather the following information as part of this research: Background and software engineering related experiences.

We will not allow anyone to access the information, except people directly involved in conducting the research. We will only use the information for the purposes of the research described in this form.

The information gathered will be anonymous. That means that it will not be possible to make a link between you and the information you provide.

We will protect the information by keeping it securely on encrypted file systems or password protected email accounts for a minimum of 5 years following completion of research project.

We intend to publish the results of the research. However, it will not be possible to identify you in the published results.

We will destroy the information five years after the end of the study.

F. CONDITIONS OF PARTICIPATION

You do not have to participate in this research. It is purely your decision. If you do participate, you can stop at any time. Any information provided before withdrawing will be deleted and excluded from our analysis. However, there is no way of withdrawing your responses once they are confirmed and submitted.

G. PARTICIPANT'S DECLARATION

I have read and understood this form. I have had the chance to ask questions and any questions have been answered. I agree to participate in this research under the conditions described. By clicking "Next" in this Google Form I indicate that I am at least 18 years old, have read and understood this form and agree to participate in this research study.

If you have questions about the scientific or scholarly aspects of this research, please contact the researcher. Their contact information is at the beginning of this section. You may also contact their faculty supervisor.

If you have concerns about ethical issues in this research, please contact the Manager, Research Ethics, Concordia University, 514.848.2424 ex. 7481 or oor.ethics@concordia.ca.

Part I: Background

1. **What is your gender** *

   Mark only one oval.

   - ◯ Male
   - ◯ Female
   - ◯ Prefer not to answer
   - ◯ Other: ____________________

2. **When is your birth year?** *

   Mark only one oval.

   - ◯ Before 1966
   - ◯ 1966-1976
   - ◯ 1977-1994
   - ◯ After 1994

3. **Which of the following best describe(s) you? (Check all that apply)** *

Check all that apply.

- ☐ Researcher
- ☐ Business Analyst
- ☐ Database Administrator
- ☐ Database Developer
- ☐ Full Stack Developer
- ☐ Graduate Student
- ☐ Product Owner
- ☐ Software Designer
- ☐ Software Developer
- ☐ Software Project Manager
- ☐ Software Quality Assurance Engineer
- ☐ Software Tester
- ☐ Undergraduate Student
- ☐ Other: ____________________

4. **How many years of professional experience do you have as a software engineer?** *

Mark only one oval.

- ○ 0
- ○ <1 year
- ○ 1-2 years
- ○ 2-5 years
- ○ 5-10 years
- ○ 10-20 years
- ○ >20 years

5. **What programming languages do you use (or have used) in your development or maintenance tasks? (Check all that apply)** *

Check all that apply.

- ☐ Java
- ☐ C
- ☐ Python
- ☐ C++
- ☐ Visual Basic .NET
- ☐ C#
- ☐ PHP
- ☐ Javascript
- ☐ SQL
- ☐ Objective-C
- ☐ Delphi/Object Pascal
- ☐ Ruby
- ☐ MATLAB
- ☐ Other: ____________________

6. **How many open source or commercial software projects have you worked on?** *

Mark only one oval.

- ◯ 0
- ◯ 1
- ◯ 2-5
- ◯ >5

7. **How many years have you been contributing to open source (in any way)?** *

Mark only one oval.

- ◯ Never
- ◯ <1 year
- ◯ 1-2 years
- ◯ 2-5 years
- ◯ 5-10 years
- ◯ 10-20 years
- ◯ >20 years

Part II: Software Engineering Information Resources

8. **Rank the type of developer knowledge that you prefer to use to get help in your software engineering tasks?** *

Non-digital (e.g., telephone, face2face, project workbook, documents, books); Digital (e.g., Slashdot, Sourceforge, Visual Studio documentation, Eclipse documentation, mailing lists, emails); Social media (e.g., blogs, Twitter, LinkedIn, YouTube, Vimeo, Stack Overflow, Slack, Facebook, GitHub)

Mark only one oval per row.

	Highly preferable	Moderately preferable	Somewhat preferable	Not very preferable	Not preferable at all	Not sure
Non-digital	◯	◯	◯	◯	◯	◯
Digital	◯	◯	◯	◯	◯	◯
Social media	◯	◯	◯	◯	◯	◯

9. **Indicate if your organization has blocked access to any social media and specify the name of the blocked social media? (e.g., Twitter, Facebook, YouTube, Vimeo, LinkedIn, Slack, GitHub, etc.)** *

10. **How important are the following social media resources for you in *completing your software engineering tasks*?** *

Mark only one oval per row.

	Very important	Moderately important	Somewhat important	Not very important	Not important at all	Not sure
Blogs	◯	◯	◯	◯	◯	◯
Facebook	◯	◯	◯	◯	◯	◯
GitHub	◯	◯	◯	◯	◯	◯
Google Groups	◯	◯	◯	◯	◯	◯
LinkedIn	◯	◯	◯	◯	◯	◯
Slack	◯	◯	◯	◯	◯	◯
Stack Overflow	◯	◯	◯	◯	◯	◯
Twitter	◯	◯	◯	◯	◯	◯
Vimeo	◯	◯	◯	◯	◯	◯
YouTube	◯	◯	◯	◯	◯	◯

11. **Rank your level of preference in using the following media as an information source to *learn new skills/concepts*?** *

Mark only one oval per row.

	Highly preferable	Moderately preferable	Somewhat preferable	Not very preferable	Not preferable at all	Not sure
Books	◯	◯	◯	◯	◯	◯
Blog posts	◯	◯	◯	◯	◯	◯
Images (e.g., Flowcharts, visualised methodologies, etc.)	◯	◯	◯	◯	◯	◯
Online written documentations	◯	◯	◯	◯	◯	◯
Podcasts	◯	◯	◯	◯	◯	◯
Question and answer sites	◯	◯	◯	◯	◯	◯
Videos	◯	◯	◯	◯	◯	◯

12. **For each task, indicate the media format that you prefer to use as a documentation. (Check all that apply)** *

Textual (e.g., blogs, Q&As, documents, books). Multimedia (e.g., videos, images, podcasts)
Check all that apply.

	Multimedia	Textual
Bug fixing	☐	☐
Finding workarounds for other problems	☐	☐
Get development tips and tricks	☐	☐
Get familiar with a software application's user interface	☐	☐
Learn a programming language	☐	☐
Learn how to customize open source projects' source code	☐	☐
Learn how to set up an application	☐	☐
Learn how to use specific features of a software application	☐	☐
Learn new concepts	☐	☐
To find answers to technical questions	☐	☐
To learn and improve my skills	☐	☐

Part III: On the Use of How-to Tutorial Videos in Software Engineering Domain

13. **Is YouTube accessible in your organization/country?** *

Mark only one oval.

◯ Yes
◯ No
◯ Yes, but filtering is bypassed using other methods

14. **For what software engineering-related purpose(s) do you watch tutorial videos? (Check all that apply)** *
Check all that apply.

- ☐ Bug fixing
- ☐ Finding workarounds for other problems
- ☐ Get development tips and tricks
- ☐ Get familiar with a software application's user interface
- ☐ Learn a programming language
- ☐ Learn how to customize open source projects' source code
- ☐ Learn how to set up an application
- ☐ Learn how to use specific features of a software application
- ☐ Learn new concepts
- ☐ Other: ______

15. **How many tutorial videos have you created in the past to share your software development knowledge?** *
Mark only one oval.

- ◯ 0
- ◯ 1
- ◯ 2-5
- ◯ 5-10
- ◯ >10
- ◯ Other: ______

16. **For what software engineering-related purpose(s) have you created tutorial videos? (Check all that apply)**
Check all that apply.

- ☐ Bug fixing
- ☐ Demonstrate a software application's user interface
- ☐ Give development tips and tricks
- ☐ Teach a programming language
- ☐ Teach how to set up an application
- ☐ Teach how to use specific features of a software application
- ☐ Teach new concepts
- ☐ Teach workarounds for other problems
- ☐ I have never created a tutorial video myself
- ☐ Other: ______

17. **How many tutorial videos have you created to share your expertise in the use of features found in a software application?**

Mark only one oval.

- ◯ 0
- ◯ 1
- ◯ 2-5
- ◯ 5-10
- ◯ >10
- ◯ Other: ____________________

18. **How often do you watch tutorial videos as a learning resource?** *

Mark only one oval.

- ◯ Daily
- ◯ Weekly
- ◯ Monthly
- ◯ Almost never
- ◯ Never

19. **How often are you unable to find a tutorial video that clearly describes what you need?** *

Mark only one oval.

- ◯ Always
- ◯ Often
- ◯ Sometimes
- ◯ Rarely
- ◯ Never

20. **To what extent do you find tutorial videos an effective learning tool in the software engineering domain?** *

Mark only one oval.

- ◯ Very effective
- ◯ Moderately effective
- ◯ Somewhat effective
- ◯ Not very effective
- ◯ Not effective at all
- ◯ Not sure

21. **To what extent do you find the information presented in a tutorial video useful for your tasks? ***

Mark only one oval.

- ⬭ Very useful
- ⬭ Moderately useful
- ⬭ Somewhat useful
- ⬭ Not very useful
- ⬭ Not useful at all
- ⬭ Not sure

22. **How often are you satisfied with the audio and visual quality of a tutorial video? ***

Mark only one oval.

- ⬭ Always
- ⬭ Often
- ⬭ Sometimes
- ⬭ Rarely
- ⬭ Never

23. **Indicate how important the following information is when selecting a tutorial video to watch? ***

Mark only one oval per row.

	Very important	Moderately important	Somewhat important	Not very important	Not important at all	Not sure
Author	⬭	⬭	⬭	⬭	⬭	⬭
Content (by watching part of the video)	⬭	⬭	⬭	⬭	⬭	⬭
Content of the speech transcription	⬭	⬭	⬭	⬭	⬭	⬭
Description	⬭	⬭	⬭	⬭	⬭	⬭
Having a narrator	⬭	⬭	⬭	⬭	⬭	⬭
HD Quality	⬭	⬭	⬭	⬭	⬭	⬭
Long video	⬭	⬭	⬭	⬭	⬭	⬭
Number of likes	⬭	⬭	⬭	⬭	⬭	⬭
Number of views	⬭	⬭	⬭	⬭	⬭	⬭
Short video	⬭	⬭	⬭	⬭	⬭	⬭
Textual annotations in the video	⬭	⬭	⬭	⬭	⬭	⬭
Title	⬭	⬭	⬭	⬭	⬭	⬭

24. **From your experience, what would you consider some of the benefits of using tutorial videos compared to written documentation? ***

25. **From your own experience, are there any disadvantages in using tutorial videos over written documentation?** *

26. **How important is having a narrator for tutorial videos?** *
Mark only one oval.

- ◯ Very important
- ◯ Moderately important
- ◯ Somewhat important
- ◯ Not very important
- ◯ Not important at all
- ◯ Not sure

27. **How often do you turn on closed captioning (subtitles) while watching a tutorial video?** *
Closed captioning or subtitle is the processes of displaying text on video screen, or other visual display. It is typically used as a transcription of the audio portion of a program as it occurs.
Mark only one oval.

- ◯ Always
- ◯ Often
- ◯ Sometimes
- ◯ Rarely
- ◯ Never

28. **Assume that you are new to WordPress and you want to know "how to add a blog post to WordPress". Having the choice between Stack Overflow, the official online documentation, and a tutorial video, all containing the same relevant information, which one will you use to learn how to perform the task?** *
Mark only one oval.

- ◯ Official online document
- ◯ Stack Overflow Q&A
- ◯ Tutorial video
- ◯ All of the above
- ◯ Other:

29. **Assume that you are new to WordPress and you want to know "How to restrict WordPress site access by IP". If you find the relevant question and answer on Stack Overflow, the official online documentation, and a tutorial video, which one will you use to learn how to perform the task?** *
Mark only one oval.

- ◯ Official online documentation
- ◯ Stack Overflow Q&A
- ◯ Tutorial video
- ◯ All of the above
- ◯ Other: ______________

Part IV: How-to Tutorial Videos as a Software Artifact

Think about an open source project which you would like to contribute to and do not have any experience in. With this assumption in mind, please answer the following questions.

30. **How often do you decide to watch a video of a software project whose source code you want to customize?** *
Mark only one oval.

- ◯ Always
- ◯ Often
- ◯ Sometimes
- ◯ Rarely
- ◯ Never

31. **Have you ever tried to locate the source code artifacts of a feature that is being demonstrated in a video?** *
Mark only one oval.

- ◯ Always
- ◯ Often
- ◯ Sometimes
- ◯ Rarely
- ◯ Never

32. **Which of the following use cases of recommending videos do you find interesting? (Check all that apply)** *
Check all that apply.

- ☐ Videos relevant to bug reports
- ☐ Videos relevant to build configurations
- ☐ Videos relevant to online textual documentation
- ☐ Videos relevant to security issues
- ☐ Videos relevant to Stack Overflow question and answers
- ☐ Other: ______________

33. **Please share any suggestion about the role or usefulness of how-to tutorial videos in software engineering domain.** *

Powered by
Google Forms